КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття ступеня магістра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

КЛАСИФІКАЦІЯ МНОЖИН ВЕКТОРІВ НА ОСНОВІ СКІНЧЕННИХ АВТОМАТІВ

Виконав студент 2-го курсу магістратури
Олександр ПАРХОМЕНКО _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Сергій КРИВИЙ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем «____» _____ 2021 р., протокол № ___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

АП – арифметика Пресбургера;

ДНК – дезоксирибонуклеїнова кислота;

СА – скінченний автомат;

ДСА – детермінований скінченний автомат;

НДСА – недетермінований скінченний автомат;

НДСА-ДСА – алгоритм для перетворення недетермінованого скінченного автомата в детермінований скінченний автомат;

НЕР-НДСА – алгоритм для перетворення нерівності в недетермінований скінченний автомат;

РІВ-НДСА – алгоритм для перетворення рівності в недетермінований скінченний автомат;

РЕФЕРАТ

Обсяг роботи 52 сторінок, 12 ілюстрацій, 5 таблиці, 6 джерел посилань.

ЗАДАЧА КЛАСИФІКАЦІЇ. АВТОМАТНИЙ МЕТОД. КЛАСИФІКАЦІЯ ГІПЕРПЛОЩИН. АРИФМЕТИКА ПРЕСБУРГЕРА

Об’єктом роботи є процес класифікації векторів відносно гіперплощини з використанням автоматного методу.

Предметом роботи є програмний засіб для класифікації векторів відносно гіперплощини з використанням автоматичного методу.

Метою роботи є створення програмного засобу для класифікації векторів відносно гіперплощини на основі автоматного методу.

Інструменти розроблення: мова програмування Python

Результати роботи: виконано загальний огляд методів вирішення задачі класифікації, проаналізовано переваги та недоліки цих методів, дослідження застосування автоматичного підходу для вирішення задачі класифікації, розроблено програмний засіб для класифікації векторів відносно гіперплощини з використанням автоматного методу.

ВСТУП

Лінійні обмеження, або лінійні констрейнти відіграють важливу роль в різноманітних застосуваннях. Важливість цих обмежень пояснюється тим, що до їх розв'язання зводиться велика кількість задач алгебри, геометрії, теорії чисел, математичного аналізу, оптимізації тощо. А в науках про обчислення ці обмеження відіграють особливу роль в контексті проблем автоматизації пошуку доведення теорем у формальних логічних мовах та верифікації програмного забезпечення. Тут виникають системи лінійних обмежень над так званими дискретними областями, до яких відносяться множини цілих, натуральних чисел та скінченні кільця і поля. В останнє десятиліття, активно вивчаються і розвиваються методи дослідження властивостей реактивних систем, тобто систем, які на відміну від функціональних, повинні працювати потенціально нескінченний час.

Прикладами такого типу систем є мобільні телефони, операційні системи, системи підготовки та редакції текстів.

Аналіз таких систем вимагає розробки спеціальних методів верифікації, спрямованих на перевірку виконуваності очікуваних властивостей систем. Ці властивості, як правило, подаються у вигляді формул деякої логічної мови, виконуваність яких перевіряється на математичній моделі реальної системи або доводяться як теореми в цій логічній мові. Аналіз сучасного стану в області існуючих математичних моделей показує, що очевидну перевагу в різноманітних застосуваннях надають автоматним і мережевим моделям. Популярність автоматних і мережевих моделей пояснюється тим, що для цих моделей багато корисних властивостей є алгоритмічно розв'язуваними, а звідси випливає можливість автоматизації більшості етапів аналізу таких властивостей. Важливими властивостями мережевих моделей є їх здатність моделювати не тільки послідовні, але й паралельні процеси.

Важливими застосуваннями методів розв'язання систем лінійних обмежень є перевірка виконуваності формул арифметики Пресбургера, розпаралелювання циклів послідовних програм, пошук найменшої суперечної підмножини диз'юнктів в заданій множині диз'юнктів. Формула арифметики Пресбургера может бути одни з способів представлення гіперплощин. [1]

Дана робота присвячена задачі класифікації векторів відносно гіперплощин. Як правило для вирішення такої задачі використовують інструменти статистики і машинного навчання, у свою чергу дана робота присвячена використанню скінченних автоматів для вирішення задачі класифікації.

1. ОСНОВНІ ВІДОМОСТІ

1.1 Поняття класифікації та її застосування

На сьогодні задача класифікації, або її також називають задачею розпізнавання, набуває все більшого значення. Це пов’язано в першу чергу з поширенням використання комп’ютеризованих систем для ідентифікації та валідації даних. Окрім того, класифікація є однією з найважливіших задач Data Mining (також глибинний аналіз даних).

Класифікація даних має безліч видів і знаходить своє застосування у багатьох сферах. Насамперед вона застосовується у сфері фінансів, у банківський та торговельних сферах. Розпізнавання зображень застосовується для перевірки підписів, ідентифікації людських облич по фотографії, навіть у військовій сфері, наприклад для розпізнавання наземних об’єктів безпілотними літальними апаратами.

Окрім цього існує безліч інших прикладів застосування класифікації:

- Комп'ютерне бачення
- Медична візуалізація та аналіз медичних зображень
- Оптичне розпізнавання символів
- Відстеження відео
- Відкриття та розробка ліків
- Токсикогеноміка
- Кількісні структурно-діяльнісні відносини
- Геостатистика
- Розпізнавання мови
- Розпізнавання рукописного тексту
- Біометрична ідентифікація
- Біологічна класифікація
- Статистична обробка природної мови
- Класифікація документів

- Інтернет- пошукові системи
- Кредитний бал
- Розпізнавання образів
- Рекомендаційна система
- Класифікація мікромасивів

Загалом задачу класифікації можна визначити як задачу, в якій, на основі множини вхідних об’єктів, що мають певний поділ на класи, будується система, яка буде спроможна класифікувати об’єкти на класи, що визначені вхідними даними. Тобто, класифікувати об’єкт означає — визначити клас, до якого належить цей об’єкт.

Для проведення класифікації за допомогою математичних методів необхідно мати формальний опис об'єкта, яким можна оперувати, використовуючи математичний апарат класифікації. Таким описом найчастіше виступає база даних. Кожний запис бази даних несе інформацію про деяку властивість об'єкта.

Як правило задачу класифікації відносять до проблем машинного навчання і вирішують із застосуванням методу штучних нейронних мереж. Проте існують і інші підходи до вирішення цієї задачі, зокрема підхід, що базуєтсья на використанні теорії автоматів. Такий підхід, наприклад, використовуєтсья у біології та генетиці для розпізнавання властивостей існуючих сполук, зокрема, розпізнаються властивості молеукл ДНК. Ці застосування пов’язані і спеціальними мовами, які акцептуються скінченними автоматами. [2]

1.2 Загальноприйняті підходи до вирішення задачі класифікації.

Оскільки проблема класифікації не нова, очевидно, що існують способи вирішення даної проблеми. Важливо розуміти їх основні принципи, переваги та недоліки.

Логістична регресія

Логістична регресія - це алгоритм машинного навчання для класифікації. У цьому алгоритмі ймовірності, що описують можливі результати одного випробування, моделюються за допомогою логістичної функції.

Серед переваг можна виділити те, що логістична регресія є найбільш корисною для розуміння впливу декількох незалежних змінних на одну змінну результату.

Суттєвим недоліком є обемеження: даний метод працює лише тоді, коли передбачувана змінна є двійковою та передбачається, що всі предиктори не залежать один від одного і дані не містять відсутніх значень.

Наївний Байєс

Наївний алгоритм Байєса, заснований на теоремі Байєса з припущенням незалежності між кожною парою ознак. Наївні класифікатори Байєса добре працюють у багатьох реальних ситуаціях, таких як класифікація документів та фільтрація спаму.

Серед переваг можна виділити те, що цей алгоритм вимагає невеликої кількості навчальних даних для оцінки необхідних параметрів. Наївні класифікатори Байєса надзвичайно швидкі в порівнянні з більш досконалими методами.

Стохастичний градієнтний спуск

Стохастичний градієнтний спуск - це простий і дуже ефективний підхід до лінійних моделей. Такий підхід особливо корисний, коли кількість зразків дуже велика.

Основною перевагою є ефективність та простота впровадження. А серед недоліків варто виділити те, що потрібен ряд гіперпараметрів і він чутливий до масштабування функцій.

Метод K-найближчих сусідів

Класифікація на основі сусідів - це тип лінивого навчання, оскільки він не намагається побудувати загальну внутрішню модель, а просто зберігає екземпляри навчальних даних. Класифікація обчислюється з простої більшості голосів k найближчих сусідів кожного пункту.

Перевагою є те, що цей алгоритм простий у впровадженні, надійний до шумних навчальних даних та ефективний, якщо навчальні дані великі.

Основний недолік полягає в тому, що потрібно визначити значення K, а вартість обчислення висока, оскільки потрібно обчислити відстань кожного екземпляру до всіх навчальних зразків.

Існують ще й інші підходи, вони, як і згадані, перш за все відносяться до машинного навчання. Як бачимо такі підходи мають свої переваги та свої недоліки. На противагу підходам машинного навчання, автоматний підхід не вимагає великої кількості вхідний даних.

1.3 Арифметика Пресбургера (АП)

Арифметика Пресбургера є логічною мовою для формулювання властивостей цілих чисел, які виражаються лінійними обмеженнями, побудованими за допомогою бінарної операції додавання $(+)$ та бінарних предикатів рівності $(=)$ і нерівності $(\leq)$. Типовим прикладом таких обмежень є $"x + y - 2z \leq 3"$ і $"2x - y + z = 3"$. Ці обмеження виконуються на деяких значеннях (n_x, n_y, n_z) невідомих (x, y, z), наприклад, трійка значень $(2, 3, 2)$ виконує обидві вищенаведені формули, а трійка $(4, 2, 1)$ не виконує жодної. Ідея використання СА для перевірки виконуваності лінійних обмежень виникла ще на початку 60-х років минулого століття в роботах Бюхі [3], Черча [4] і Елгота [5]. Основним тут був існуючий зв'язок між атомарним лінійним обмеженням і автоматом,

який акцептує всі моделі цього обмеження (тобто всі розв'язки даного обмеження). Тоді, використовуючи замкнутість булевих операцій на множині мов, які акцептуються СА, можна побудувати автомат, який акцептує всі розв'язки довільної заданої формули АП. В означенні АП використовується алфавіт змінних $X = \{x, y,..., z, x_1, y_2,...\}$, дві константи 0 і 1 (як нульарні операції) і бінарна операція додавання $+$.

Означення 1. *Термом АП називається:*

- *довільна змінна із X і константи 0, 1;*
- *якщо* t_1 *і* t_2 *терми АП, то* $t_1 + t_2$ *терм АП.*

При зображенні термів використовуються звичайні скорочення, прийняті в арифметиці. Так терм $x + y + y + z + z + z + 1 + 1$ записується в скороченому вигляді як $x + 2y + 3z + 2$.

Означення 2. *Атомарною формулою АП називається рівність або нерівність між двома термами, тобто вирази* $t = t'$ *або* $t \leq t'$*, де* $t, t' -$ *терми АП.*

Наприклад, вирази $x + 2y = 3z + 1$ і $2x + 1 \leq y$ є атомарними формулами АП. Перша з цих формул може бути записана як $x + 2y - 3z - 1 = 0$.

Означення 3. *Формулою АП називається*

- *довільна атомарна формула АП;*
- *якщо* C*і* C' *формули АП, то* $C \vee C'$, $C \wedge C'$ *формули АП;*
- *якщо* C *формула АП, то* $\neg C, \exists x$ *і* $\forall x\ C\ -$ *формули АП.*

Вільні і зв'язані змінні у формулах АП визначаються стандартним способом: змінна x називається зв'язаною у формулі C, якщо вона знаходиться в області дії квантора існування або загальності. В протилежному випадку змінна називається вільною. Областю інтерпретації формул АП є множина натуральних чисел $\mathcal{N}$, в якій символи $0, 1, +, =, \leq$ мають свій звичайний сенс.

Розв'язком або моделлю формули $C(x_1, x_2, ..., x_q)$ називається вектор $(c_1, c_2, ..., c_q) \in \mathcal{N}$, який задовольняє формулу C (тобто формула $C(c_1, c_2, ..., c_q)$ істинна). Наприклад, вектор $(0, 2, 1)$ розв'язок формули $x + 2y = 3z + 1$. Якщо для заданої формули АП $C(x_1, x_2, ..., x_q)$ існує розв'язок, то говорять, що ця *формула виконується*.

2. ТЕОРЕТИЧНА БАЗА

2.1 Виконуваність формул АП

Нехай $C(x_1, x_2,..., x_q)$ − атомарна формула АП, вільні змінні якої належать множині $X = \{x_1, x_2,..., x_q\}$. Розглянемо питання про побудову скінченного автомата $\boldsymbol{\mathcal{A}} = (A, X, f, a_0, F)$, який акцептує розв'язки формули C. При побудові автомата використовується двійкове зображення натуральних чисел, тобто у вигляді двійкових слів в алфавіті $\{0, 1\}$. Наприклад, число 13 в такому поданні є двійковим словом 1011, а пара $(13, 6)$ набуває вигляду

$$\begin{matrix} 1011 \\ 0110 \end{matrix}.$$

Символи вхідного алфавіту автомата − це порозрядні двійкові вектори, з яких будується двійкове зображення n-ки натуральних чисел. Так, для пари $(13, 6)$ символами алфавіту будуть вектористовпчики

$$z_1 = [1, 0]^T,\ z_2 = [0, 1]^T,\ z_3 = [1, 1]^T,\ z_4 = [1, 0]^T,$$

де $[n_x, n_y]^T$ означає транспонування вектора $[n_x, n_y]$. Користуючись цим зображенням, будується алгоритм, який для довільної атомарної формули АП, яка є лінійним обмеженням

$$a_1x_1 + a_2x_2 + ... + a_qx_q \le b\,,$$

будує скінченний *X*-автомат, що акцептує розв'язки цього обмеження. Це обмеження записується в скороченому вигляді $(a, x) \le b$, де (a, x) означає

скалярний добуток векторів $a = (a_1, a_2,..., a_q)$ і $x = (x_1, x_2,..., x_q)$. Побудова автомата відбувається таким чином.

Нехай перед прочитанням вхідного символа z_i автомат знаходився в стані s, а після його прочитання – в стані s'. Тоді отримуємо такі значення для цих станів: $s = [1/2^i (b - (a, c^i))]$ і $s' = [1/2^{i+1}(b - (a, c^{i+1}))]$. За означенням $c^i = (\sum_{j=0}^{i-1} 2^j z_{1j},..., \sum_{j=0}^{i-1} 2^j z_{qi})$ і тому $c^{i+1} = c^i + 2^i z_i$. Підставляючи $c^i + 2^i z_i$ замість c^{i+1} у вираз для s', знаходимо

$$s' = [1/2^{i+1}(b - (a, c^i) - 2^i (a, z_i))] \quad \text{або}$$

$$2s' = [1/2^i (b - (a, c^i) - (a, z_i))] = [s - (a, z_i)]$$

Звідки випливає, що $s' = [1/2(s - (a, z_i))]$. Отже, для кожного стану s і стовпчика $z_i \in \{0, 1\}^q$ визначаємо функцію переходів

$$f(s, z_i) = [1/2(s - (a, z_i))] = s'.$$

Початковим станом є b, а заключними станами будуть ті, які мають значення більші або рівні 0, оскільки із (1) маємо $b - (a_1 x_1 + a_2 x_2 + ... + a_q x_q) \geq 0$.

Зі сказаного випливає алгоритм АП-ДСА побудови СА, який акцептує всі розв'язки лінійного обмеження $(a, x) \leq b$.

АП-ДСА (φ)

Вхід: атомарна формула $\varphi = ((a, x) \leq b)$ *над* $\mathcal{N}$

Вихід: ДСА $A_\varphi = (A, X, f, a_0, F)$ *, який акцептує всі розв'язки формули* φ

Метод:

$A, f, F = \emptyset$; $a_0 \leftarrow \{b\}$

$W = \{b\}$

while $W \neq \emptyset$ do:

pick s from W

add s to A

if $s \geq 0$ then add s to F

for all $z \in \{0, 1\}^q$ do:

$j = [1/2(s - (a, z_i))]$

if $j \notin A$ then add j to W

add (s, z, j) to f

return (A, X, f, a_0, F)

Часткова правильність алгоритму АП-ДСА доводиться методом математичної індукції за довжиною вхідного слова. Дійсно, для довільного стану $k \in \mathbb{Z}$ і довільного слова $p \in (\{0, 1\}^q)^*$ цей стан акцептує слово p, якщо воно є двійковим кодом такого $c \in \mathbb{N}^q$, яке задовольняє обмеження $\varphi = ((a, c) \leq k)$.

Для $l(p) = 0$ справедливість твердження випливає безпосередньо з означення заключних станів автомата. А для слів ненульової довжини це випливає з припущення індукції і того, що функція переходів f задовольняє обмеження $(a, c) \leq k$. Термінальність алгоритму АП-ДСА дає наступна лема:

Лема 1. *Нехай* $\varphi = ((a,\ c) \leq b)$ *і* $m = \sum_{i=1}^{q} |a_i|$, *тоді стани j, які породжуються в процесі виконання алгоритму АП-ДСА, задовольняють нерівностям* $-|b| - m \leq j \leq |b| + m$.

Розглянемо тепер обмеження типу $\varphi = ((a, x) = b)$. З властивостей станів автомата, які розглядалися в алгоритмі АП-ДСА, випливає, що для цього обмеження початковим станом, як і раніше, буде стан b, а заключним станом буде 0. Крім того, наступний стан s' має бути таким, щоб рівняння $(a, c') = 1/2(s - (a, z))$ мало розв'язок. А це буде в тому випадку, коли число $s - (a, z)$ буде парним, в протилежному випадку перехід зі стану s в стан s'невизначений. Отже, алгоритм АПР-ДСА для обмеження $(a, x) = b$ набуває вигляду:

АПР-ДСА (φ)
Вхід: атомарна формула $\varphi = ((a, x) = b)$ *над* $\boldsymbol{\mathcal{N}}$
Вихід: ДСА $A_\varphi = (A, X, f, a_0, F)$, *який акцептує всі розв'язки формули* φ
Метод:

$A, f, F = \emptyset;\ a_0 \leftarrow \{b\}$
$W = \{b\}$
while $W \neq \emptyset$ do:
    pick s from W
    add s to A
    if $s = 0$ then add s to F
    for all $z \in \{0, 1\}^q$ do:
        if $(s - (a, z))$ is even then
            $j = 1/2(s - (a, z_i))$
            if $j \notin A$ then add j to W
            add (s, z, j) to f

od

od

od

return (A, X, f, a_0, F)

Побудований автомат буде частковим і в разі наобхідності його можна перетворити до повного автомата. А правильність алгоритму АПР-ДСА випливає з такої теореми.

Теорема 1. *Алгоритм АПР-ДСА закінчує свою роботу і будує автомат, кількість станів якого обмежена величиною* $max(|b|, |a_1| + |a_2| + \ldots + |a_q|)$ *і який акцептує всі розв'язки обмеження* $a_1x_1 + a_2x_2 + \ldots + a_qx_q$.

2.1.1 Подання розв'язків формули $C(x_1, x_2, \ldots, x_q)$.

Розглянемо випадок, коли маємо автомат $\boldsymbol{\mathcal{A}}_{C(x_1,x_2,\ldots,x_q)}$, який акцептує розв'язки формули $C(x_1, x_2, \ldots, x_q)$, а потрібно побудувати автомат, який акцептує розв'язки формули $C(x_1, x_2, \ldots, x_q, y)$. Це означає, що у формулі $C(x_1, x_2, \ldots, x_q)$ з'явився ще один доданок вигляду $c'y$. Для цього скористаємося алгоритмом АПР-ДСА.

Додавання однієї вільної змінної веде до збільшення числа станів і числа символів вхідного алфавіта (це число подвоюється в порівнянні з числом символів у вхідному алфавіті автомата $\boldsymbol{\mathcal{A}}_{C(x_1,x_2,\ldots,x_q)}$). Якщо використовується алгоритм АПР-ДСА, то неважко збагнути, що структура автомата $\boldsymbol{\mathcal{A}}_{C(x_1,x_2,\ldots,x_q)}$ не зміниться і він входить в автомат $\boldsymbol{\mathcal{A}}_{C(x_1,x_2,\ldots,x_q,y)}$, а змінюються тільки символи вхідного алфавіту і з'являються переходи в існуючі і нові стани. Це випливає з того, що нові стани генеруються за допомогою формули

$$j = [1/2(s - a_1z_1 - a_2z_2 - \ldots - a_qz_q - a_{q+1}z_{q+1})]$$

де a_{q+1} коефіцієнт при новій вільній змінній y.

Розглянемо ще два види автоматів, які акцептують розв'язки формул вигляду $\neg C(x_1, x_2,..., x_q)$ і $\exists x_i C(x_1, x_2,..., x_q)$.

2.1.2 Подання розв'язків формули $\neg C(x_1, x_2,..., x_q)$.

Автомат, який акцептує розв'язки формули $\neg C(x_1, x_2,..., x_q)$, будується за автоматом $\mathcal{A}_{C(x_1,x_2,...,x_q)}$, виходячи із такого зауваження. Якщо мова $L \subseteq F(X)$ акцептується автоматом $\mathcal{A}_C$, то доповнення цієї мови $F(X) \backslash L$ акцептується автоматом $\mathcal{A}'_C$, який відрізняється від автомата A_C тільки множиною заключних станів, яка дорівнює доповненню множини заключних станів автомата $\mathcal{A}_C$. Отже, ніяких додаткових побудов виконувати немає потреби, якщо побудовано автомат $\mathcal{A}_C$.

2.1.3 Подання розв'язків формули $\exists x_i C(x_1, x_2,..., x_q)$.

Зауважимо, що формула $\forall x C$ логічно еквівалентна формулі $\neg \exists x \neg C$, тому розглянемо подання для формули $\exists x C$. Якщо автомат $A_{C(x_1,x_2,...,x_q)}$, який акцептує розв'язки формули $C(x_1, x_2,..., x_q)$, побудований, то побудова автомата $A_{\exists x_i C(x_1,x_2,...,x_q)}$ зводиться до вилучення координати x_i із множини символів вхідного алфавіта цього автомата. В загальному випадку автомат, отриманий в такий спосіб, не буде детермінованим.

2.2 Арифметика Пресбургера над кільцем цілих чисел

Зображення цілих чисел у вигляді *2-доповнення* дає можливість побудувати НДСА, який акцептує розв'язки лінійних обмежень над кільцем цілих чисел $\mathbb{Z}$.

2-доповнення для числа $x \in \mathbb{Z}$ визначається таким чином: для $x \geq 0$ двійкове зображення має вигляд $(x_0 x_1 \ldots x_{n-1} x_n)$, у якого $x_n = 0$, а зображення числа $-x$ у вигляді 2-доповнення за двійковим зображенням x визначається як

$$-x = (\sum_{i=0}^{n-1} \neg x_i 2^i - \neg x_n 2^n) + 1,$$

де $\neg x_n = 1$, $\neg x_i = 1 - x_i$ для $0 \leq i \leq n-1$, а $+$ є звичайною операцією додавання. Біт x_n у такому зображенні називається знаковим бітом. Наприклад, слово 110 зображує число $1 + 2 - 0 = 3$, а слово 111 зображує число $1 + 2 - 4 = -1$.

Слова довжини 1 мають лише знаковий біт і, зокрема, слово 0 зображує число 0, а слово 1 зображує число -1. Пусте слово відповідника серед цілих чисел і воно означає відсутність числа. 2-доповнення чисел не є однозначним. Дійсно, всі слова 1, 11, 111,... зображують одне і те саме число -1. Звідси дістаємо, що регулярний вираз $x_0 x_1 .. x_{n-1} x_n (x_n)^*$ зображує одне і те саме число $x \in \mathbb{Z}$. Дійсно, для $x_n = 0$ це очевидно, а для $x_n = 0$ слова $x_0 x_1 \ldots x_{n-1} 11^*$ зображують одне і те ж число, тому що

$$-2^{m+n} + 2^{m-1+n} + \ldots + 2^{n+1} = 2^n .$$

При зображенні числа у вигляді 2-доповнення роль додаткового символа # відіграє знаковий біт x_n.

2.2.1 Випадок нерівності.

Побудуємо НДСА, який акцептує цілочисельні розв'язки атомарної формули $a \cdot x \leq b$. Станами автомата, як і раніше, виступають цілі числа, причому:

стан $c \in \mathbb{Z}$ *акцептує слово-код вектора* x*, якщо* $a \cdot x \leq c$. (**)

Початковим станом буде число b, єдиним заключним станом буде спеціальний стан a_f. Всі інші стани не є заключними. При визначенні функції переходів зі стану c в стан c' можливі два випадки:

(1) якщо двійкове слово $p' \neq e$, то $z_i p'$ кодує число $2c' + z_i$, де c' закодоване словом p'(це випливає безпосередньо з означення 2-доповнення);

(2) якщо $p' = e$, то $z_i p'$ кодує число $-z_i$, оскільки в цьому випадку z_i є знаковим бітом.

У випадку (1) властивість (**) визначає стан c' такий, що $a \cdot c \leq c' \Leftrightarrow a(2c + z_i) \leq c'$. Звідки

$$c' = [1/2(c - az_i)]$$

У випадку (2) властивість (**) тільки вимагає стану c', якщо $a \cdot (-z_i) \leq c$ і коли це так, то стан c' повинен бути заключним, тобто має бути станом a_f. Отже, якщо $a \cdot (-z) \leq c$, то крім переходу, який визначений випадком (1), потрібно додати перехід $f(c, z_i) = a_f$. Таким чином, остаточно дістаємо функцію переходів НДСА:

$$f(c, z_i) = \begin{cases} \{|1/2(c - az_i)|, a_f\} & c + az_i \geq 0, \\ \{|1/2(c - az_i)|\} \end{cases}$$

Зі всього сказаного випливає такий алгоритм побудови автомата:

НЕР-НДСА (φ)

Вхід: атомарна формула $\varphi = ((a, x) \leq b)$ *над* $\mathcal{Z}$

Вихід: НДСА $\mathcal{A} = (A, X, f, a_0, F)$ *, який акцептує всі розв'язки формули* φ

Метод:

$A, f, F = \emptyset$; $a_0 \leftarrow \{b\}$

$W = \{b\}$

while$W \neq \emptyset$ do:

pick s from W

add s to A

if $s = 0$ then add s to F

for all $z \in \{0, 1\}^q$ do:

$j = [1/2(s - (a, z_i))]$

if $j \notin A$ then add j to W

add (s, z, j) to f

$c' = s + a \cdot z$

if $c' \geq 0$ then

add a_f to A and F

add (s, z, a_f) to f

od

od

return (A, X, f, a_0, F)

З наведеного алгоритму випливає, що побудований НДСА включатиме всі стани і переходи ДСА для тієї самої атомарної формули АП над множиною натуральних чисел, з деякими додатковими станами і переходами.

2.2.2 Випадок рівності.

Побудова НДСА для формули $\varphi = (ax = b)$ виконується аналогічно до побудови НДСА для нерівності. Різниця полягає лише в тому, що він матиме додаткові переходи в заключний стан a_f тоді і тільки тоді, коли $s + az_i = 0$.

Такий автомат можна побудувати для формули $\varphi = (ax = b)$, виходячи з автомата $\boldsymbol{\mathcal{A}}$, побудованого для формули $\varphi' = (ax \leq b)$. Для цього потрібно вилучити в автоматі $\boldsymbol{\mathcal{A}}$ всі переходи такі, що $s' \neq 1/2(s - az_i)$, і всі переходи в заключний стан a_f, для яких $s + az_i \neq 0$. Звідси випливає такий модифікований алгоритм побудови автомата:

РІВ-НДСА (φ)
Вхід: атомарна формула $\varphi = (a \cdot x = b)$ *над* $\boldsymbol{\mathcal{Z}}$
Вихід: НДСА $\boldsymbol{\mathcal{A}} = (A, X, f, a_0, F)$ *, який акцептує всі розв'язки формули* φ
Метод:

$A, f, F = \emptyset$; $a_0 \leftarrow \{b\}$
$W = \{b\}$
while $W \neq \emptyset$ do:
    pick s from W
    add s to A
    for all $z \in \{0, 1\}^q$ do:
        if $(s - az)$ is even then
            if $(s + az) = 0$ then add s to F
            $j = 1/2(s - (a, z_i))$
            if $j \notin A$ then add j to W
            add (s, z, j) to f

$c' = s + a \cdot z$
if $c' \geq 0$ then
add a_f to A and F
add (s, z, a_f) to f
od
od
return (A, X, f, a_0, F)

З наведеного алгоритму випливає, що побудований НДСА включатиме, як і в попередньому випадку, всі стани і переходи ДСА для тієї самої атомарної формули АП над множиною натуральних чисел, з деякими додатковими станами і переходами. Крім того, детермінізація цього автомата не приводить до експоненціального росту числа станів. У найгіршому випадку число станів в детермінованому автоматі буде вдвічі більшим, ніж в недетемінованому автоматі

2.3 Детермінізація та мінімізація автоматів

Окрім описаних вище алгоритмів в даній роботі були використані інші алгоритми роботи за автоматами. Коротко опишемо їх.

2.3.1 Детермінізація автоматів

Оскільки для роботи у полі $\mathbb{Z}$ ми вокористовуємо алгоритми НЕР-НДСА та РІВ-НДСА результатами роботи яких є недетерміновані автомати постає пиання їх детермінізації.

З теоретичної точки зору важливим є твердження, що встановлює рівносильність детермінованих і недетермінованих автоматів відносно акцептованих ними мов.

Теорема про детермінізацію. *Для довільного недетермінованого*

X-автомата $\mathcal{A}$ *існує еквівалентний йому детермінований ініціальний X-автомат* $\mathcal{B}$ *, причому, якщо автомат має n станів, то автомат B може бути побудований так, що число його станів не перевищує 2^n.*

Доведення. Нехай (A, X, Q_A, a_0) −недетермінований X −автомат і $B(A)$ −булеан множини станів A.На $B(A)$введемо відношення досяжності: множина A'' досяжна з множини A', якщо існує слово $p \in F(X)$ таке, що $A'' = \bigcup_{a \in A'} Q^*(a, p)$.

За множину станів шуканого детермінованого X-автомата $\mathcal{B}$ візьмемо множину всіх підмножин множини станів, досяжних із множини $A_0 = \{a_0\}$. Множина $A_0 = \{a_0\}$служитиме початковим станом автомата $\mathcal{B}$. Функцію переходів автомата $\mathcal{B}$ задамо за допомогою рівності $f_{\mathcal{B}}(\tilde{A}, x) = \bigcup_{a \in \tilde{A}} Q(a, x)$, де $\tilde{A}$ −стан автомата $\mathcal{B}$.

Нехай $\overline{A}$ довільна підмножина із A і нехай $\overline{B}$ −всі підмножини A досяжні із $A_0 = \{a_0\}$ і такі, що їх перетин з $\overline{A}$ не пустий. Покажемо методом виключення в обидів сторони, що мови, які акцептуються X − автоматами. $(A, X, Q_A, a_0, \overline{A})$ і $(B, X, Q_B, \{a_0\}, \overline{B})$, збігаються.

Нехай слово $p = x_1 x_2 \dots x_n$ таке, що $Q_A(a_0, p) \cap \overline{A} \neq \emptyset$. Це означає, що існує послідовність станів $a_0, a_1, \dots, a_n = a'$ така, що $a_{i+1} \in Q_A(a_i, x_{i+1})$ для $i = 0, 1, \dots, n-1$. Але тоді множина, що представляє стан $f_B(\{a_0\}, p)$, має включати стан a', а тому стан a' має належати до $\overline{B}$.Навпаки, нехай для $p = x_1 x_2 \dots x_n$ стан $f_B(\{a_0\}, p)$ належить множині $\overline{B}$. Це означає, що

множина, яка представляє цей стан, включає деякий елемент a'із $\overline{A}$. Тоді з означення досяжності множин і способу задання функції f_B індукцією за довжиною слова p можна побудувати послідовність $a' = a_n, a_{n-1},..., a_0$ таку, що $a_i \in f_B(a_0, x_1 x_2 ... x_i)$, а це означає при $i = n$, що $Q_A^*(a_0, p) \cap \overline{A} \neq \emptyset$.

Варто зауважити, що множина, яка складається з n елементів, має 2^n підмножин, і, отже, число станів автомата $\mathcal{B}$ не перевищує 2^n. ▌

З доведення теореми про детермінізацію випливає алгоритм НДСА-ДСА, що перетворює НДСА A в ДСА B, розпізнаючи ту ж мову.

НДСА-ДСА (A)

Вхід: НДСА $\mathcal{A} = (A_{\mathcal{A}}, X, f_{\mathcal{A}}, a_0, F_{\mathcal{A}})$,

Вихід: ДСА $\mathcal{B} = (A, X, f, a_0, F)$, *що акцептує ту ж мову, що НДСА* $\mathcal{A}$

Метод:

$A, f, F = \emptyset$; $a_0 \leftarrow \{b\}$

$W = \{b\}$

while $W \neq \emptyset$ do:

pick s from W

add s to A

if $s \notin F_{\mathcal{A}}$ then add s to F

for all $z \in \{0, 1\}^q$ do:

$a_f = \bigcup_{b \in s} f_{\mathcal{A}}(b, z)$

if $a_f \notin A$ then add a_f to W

add (s, z, a_f) to f

od
od
return (A, X, f, a_0, F) [4]

2.3.2 Мінімізація автоматів

Швидкість класифікації залежить від кількості станів автомату. Для кожного ДСА можна знайти еквівалентний йому ДСА з найменшим числом станів. Більш того, для довільної мови існує єдиний мінімальний ДСА. Основна ідея мінімізації ДСА полягає в тому, що поняття еквівалентності станів дозволяє об'єднувати стани в блоки таким чином.

1. Всі стану в блоці еквіваленти
2. Будь-які два стани, вибрані з різних блоків, нееквівалентні

Для мінімізації автомату використаємо алгоритм запропонований Хопфортом. [6]

Хопфорт (A)
Вхід: ДСА $\boldsymbol{\mathcal{A}} = (A, X, f, a_0, F)$,

Вихід: ДСА $\boldsymbol{\mathcal{B}} = (A, X, f, a_0, F)$

Метод:

$P = \{F, Q \backslash F\}$
$S = \emptyset$
for $c \in \{0, 1\}^q$ do
    add $\{F, c\}$, $\{Q \backslash F, c\}$ to S
while $S \neq \emptyset$ do:
    pick $\{C, a\}$ from S
    for R in P do:
        $R_1, R_2 = split(R, C, a)$

if $R_1 \neq \emptyset$ and $R_2 \neq \emptyset$ then

replace R in P with R_1 and R_2

if $\{R, c\}$ in S

remove $\{R, c\}$ from S

add $\{R_1, c\}$ to S

add $\{R_2, c\}$ to S

else

if $|P[R_1]| \leq |P[R_2]|$ then

add $\{R_1, c\}$ to S

else

add $\{R_2, c\}$ to S

return (P, X, f, a_0, F)

3. РЕАЛІЗАЦІЯ

3.1 Приклади

3.1.1 АП-ДСА

Приклад 1. Побудувати скінченний X-автомат, який акцептує розв'язки лінійного обмеження $2x - y \leq -1$.

Розв'язання. Автомат має алфавіт $z_1 = [0,0]$, $z_2 = [0,1]$, $z_3 = [1,0]$, $z_4 = [1,1]$ і початковий стан -1. Знаходимо переходи і генеруємо стани за формулою (2), яка для початкового стану -1 набуває вигляду: $f(-1,z) = [1/2(-1 - ((2,-1)(z_x, z_y)))] = [1/2(-1 - 2z_x + z_y)]$.

Отже, маємо $W = \{-1\}$, $A = \emptyset$ і

$$f(-1,[0,0]) = -1,\ f(-1,[0,1]) = 0,\ f(-1,[1,0]) = -2,$$
$$f(-1,[1,1]) = -1$$

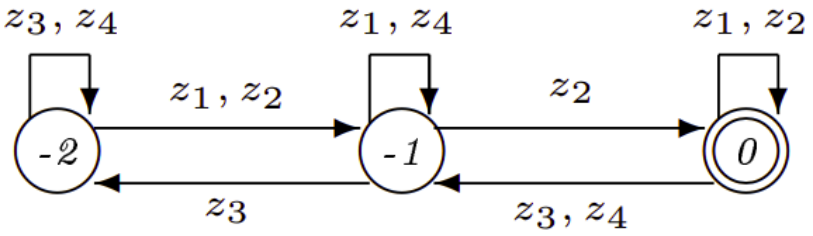
Отримуємо $W = \{0,-2\}$, $A = \{-1,0\}$, стан 0 заключний і

$$f(0,[0,0]) = 0,\ f(0,[0,1]) = 0,\ f(0,[1,0]) = -1,$$
$$f(0,[1,1]) = -1$$

Отримуємо $W = \{-2\}$, $A = \{-1,0,-2\}$ і

$$f(-2,[0,0]) = -1,\ f(-2,[0,1]) = -1,$$
$$f(-2,[1,0]) = -2,\ f(-2,[1,1]) = -2$$

Оскільки $W = \emptyset$, то на цьому робота алгоритму закінчується і дістаємо автомат $\boldsymbol{\mathcal{A}} = (A = \{-1,0,-2\}, X = \{z_1 = [0,0],\ z_2 = [0,1],\ z_3 = [1,0],\ z_4 = [1,1]\},\ f, \{-1\}, F = \{0\})$ функція переходів якого задана нижченаведеним графом переходів:



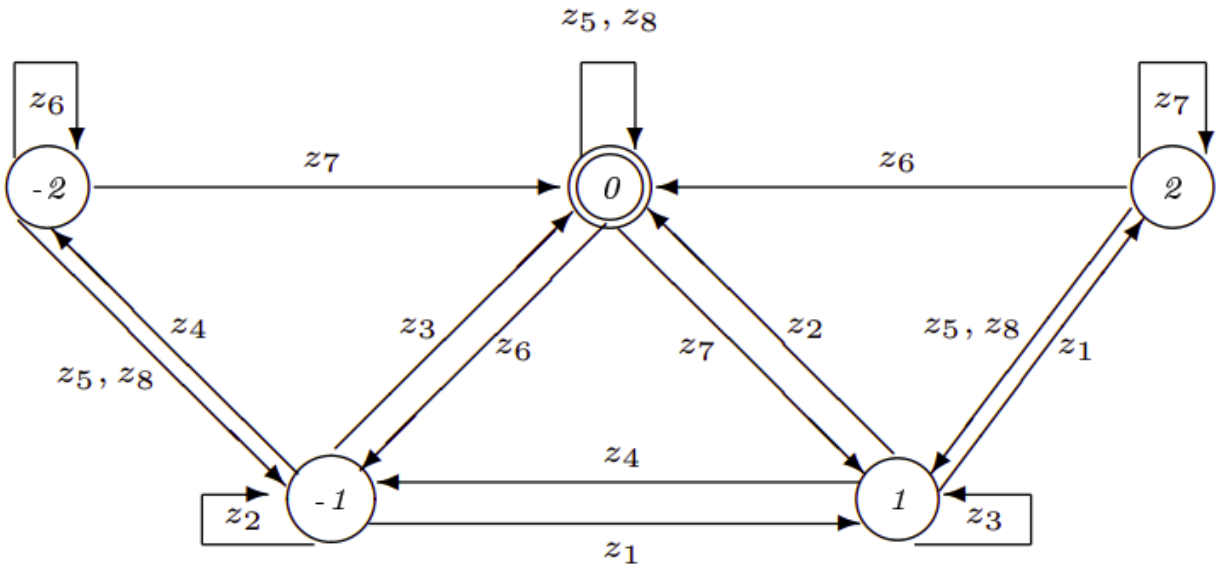
3.1.2 АПР-ДСА

Приклад 2. Проілюструємо сказане для формули АП $x + 2y + u = 3z + 1$, де u є новою вільною змінною.

Для формули $x + 2y = 3z + 1$ відповідний автомат $\boldsymbol{\mathcal{A}} = (A = \{0, 1, 2, -1, -2\}, X = \{z_1 = [0,0,1], z_2 = [1,0,0],$

$z_3 = [0,1,1], z_4 = [1,1,0], z_5 = [0,0,0], z_6 = [0,1,0],$

$z_7 = [1,0,1], z_8 = [1,1,1]\}, f, 1, \{0\})$ побудований алгоритмом АПР-ДСА, має вигляд:



Коли значення змінної u дорівнює 0, то вищенаведений автомат буде підавтоматом шуканого автомата, оскільки значення останнього

компонента в символах переходів не впливає на значення функції переходів підавтомата. Дійсно, алфавітом шуканого автомата є символи

$$z_1 = [0,0,1,0],\ z_2 = [1,0,0,0],\ z_3 = [0,1,1,0],\ z_4 = [1,1,0,0],$$

$$z_5 = [0,0,0,0], z_6 = [0,1,0,0],\ z_7 = [1,0,1,0],\ z_8 = [1,1,1,0],$$

$$z_9 = [0,0,0,1],\ z_{10} = [0,1,0,1],\ z_{11} = [1,0,1,1],\ z_{12} = [1,1,1,1],$$

$$z_{13} = [0,0,1,1], z_{14} = [1,0,0,1], z_{15} = [0,1,1,1],\ z_{16} = [1,1,0,1]$$

де перші вісім символів мають значення останнього компонента рівним 0, а перші компоненти цих символів збігаються з вхідним алфавітом підавтомата.

Застосовуючи алгоритм АПР-ДСА, генеруємо стани для початкового стану 1:

$$z_1^T = [0,0,1,0] \to j = \frac{1-0-0+3-0}{2} = 2;$$

$$z_2^T = [1,0,0,0] \to j = \frac{1-1-0+0-0}{2} = 0;$$

$$z_3^T = [0,1,1,0] \to j = \frac{1-0-2+3-0}{2} = 1;$$

$$z_4^T = [1,1,0,0] \to j = \frac{1-1-2+0-0}{2} = -1;$$

$$z_9^T = [0,0,0,1] \to j = \frac{1-0-0+0-1}{2} = 0;$$

$$z_{10}^T = [0,1,0,1] \to j = \frac{1-0-2+0-1}{2} = -1;$$

$$z_{11}^T = [1,0,1,1] \to j = \frac{1-1-0+3-1}{2} = 1;$$

$$z_{12}^T = [1,1,1,1] \to j = \frac{1-1-2+3-1}{2} = 0;$$

Для решти вхідних символів значення функції переходів невизначені. Нові стани не виникли, але з'явилися нові

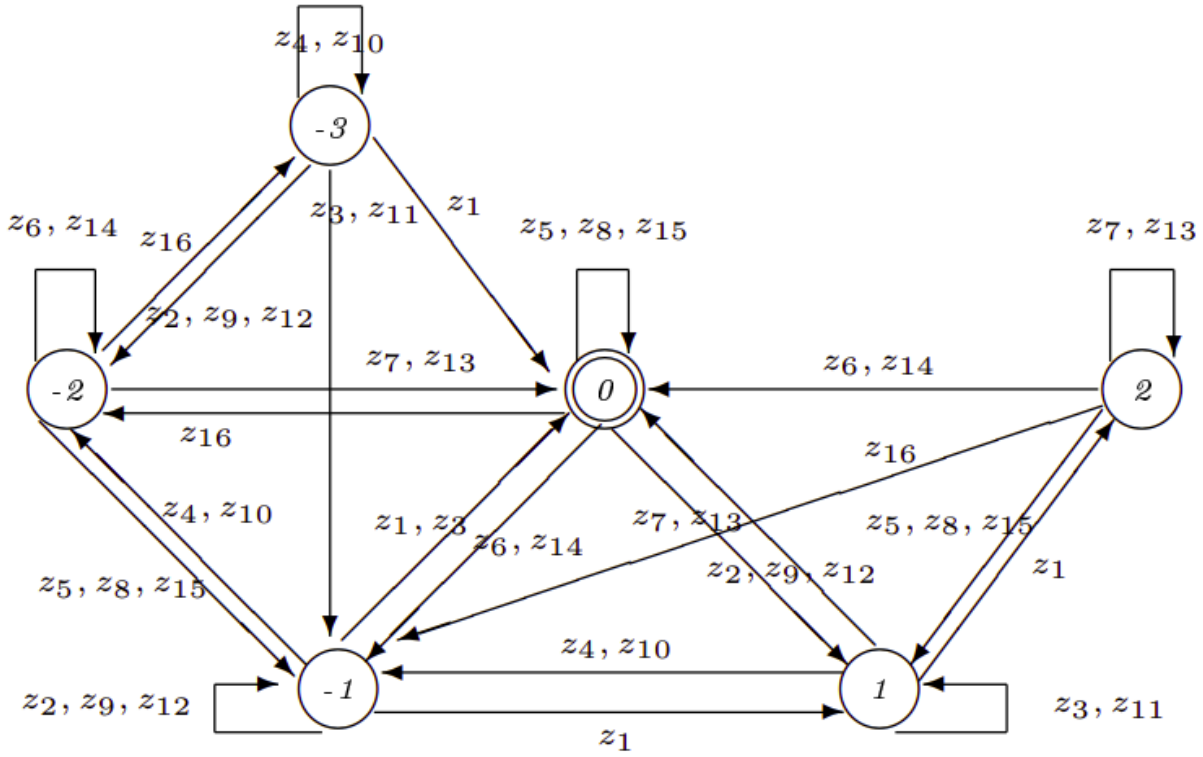
Не вдаючись в деталі обчислень нових переходів, наведемо їх остаточний вигляд:

$$1 \xrightarrow{z_2,z_9,z_{12}} 0,\ 1 \xrightarrow{z_4,z_{10}} -1,\ \ 1 \xrightarrow{z_3,z_{11}} 1,\ 1 \xrightarrow{z_1} 2.$$

Не вдаваючись в деталі обичслень нових переходів, наведемо їх остаточний вигляд:

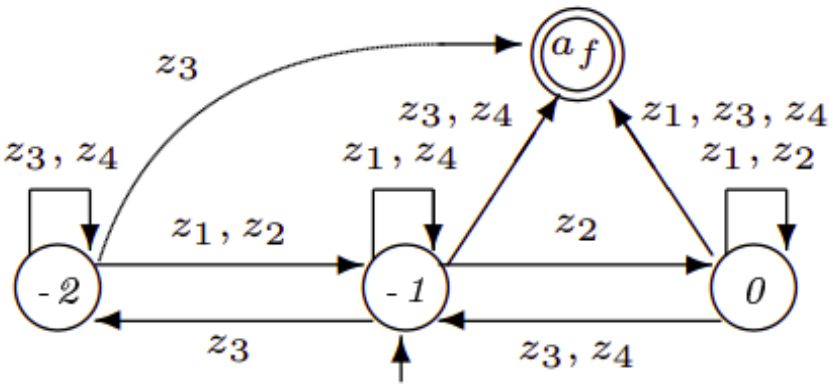
$$0 \xrightarrow{z_7,z_{13}} 1,\ 0 \xrightarrow{z_5,z_8,z_{15}} 1,\ 0 \xrightarrow{z_6,z_{14}} -1,\ 0 \xrightarrow{z_{16}} -2,$$
$$2 \xrightarrow{z_7,z_{13}} 2,\ 2 \xrightarrow{z_5,z_8,z_{13}} 1,\ 2 \xrightarrow{z_6,z_{14}} 0,\ 2 \xrightarrow{z_{16}} -1,$$
$$-1 \xrightarrow{z_2,z_9,z_{12}} -1,\ -1 \xrightarrow{z_4,z_{10}} -2,\ -1 \xrightarrow{z_3,z_{11}} 0,\ -1 \xrightarrow{z_1} 1,$$
$$-2 \xrightarrow{z_7,z_{13}} 0,\ -2 \xrightarrow{z_5,z_8,z_{13}} -1,\ -2 \xrightarrow{z_6,z_{14}} -2,\ 2 \xrightarrow{z_{16}} -3,$$
$$-3 \xrightarrow{z_2,z_9,z_{12}} -2,\ -3 \xrightarrow{z_4,z_{10}} -3,\ -3 \xrightarrow{z_3,z_{11}} -1,\ -3 \xrightarrow{z_1} 0,$$

Як бачимо, новий стан -3 з'являється при генерації переходів зі стану -2. Шуканий автомат має вигляд:



3.1.3 НЕР-НДСА

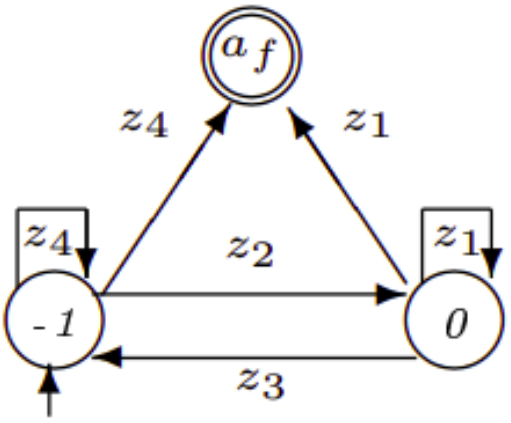
Приклад 3. Розглянемо формулу АП $2x - y \leq -1$ з прикладу 1. НДСА для цієї формули має вигляд:



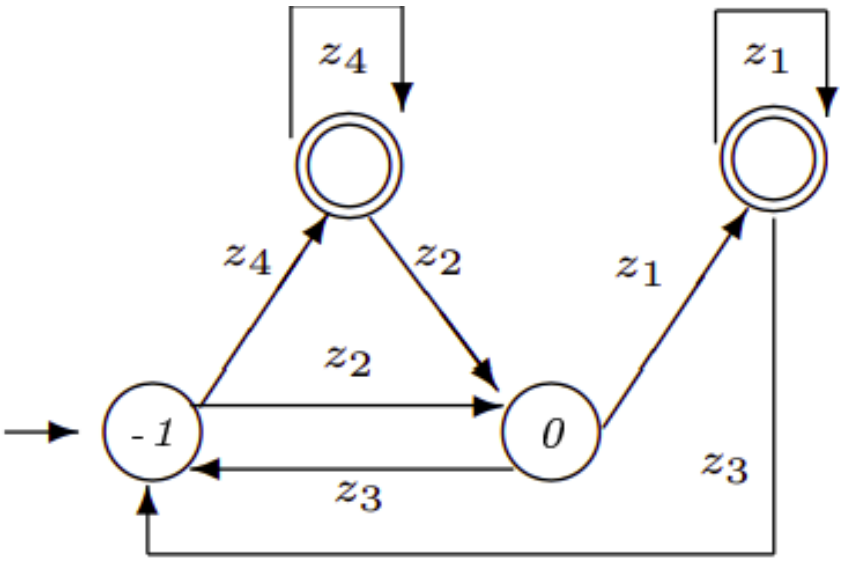
Як видно з графа переходів автомата, всі стани і переходи автомата з прикладу 1 присутні в цьому автоматі.

3.1.4 РІВ-НДСА

Приклад 4. Якщо розглядати НДСА, побудований в попередньому прикладі, то вилучаючи з нього надлишкові переходи, дістаємо такий автомат:



Детермінізація отриманого автомата приводить до такого ДСА:



3.2 Оцінка швидкодії

Найважливішим аспектом даної роботи є оцінка швидкодії обраного способу класифікації. Для цього було проведено тестування на випадково згенерованих гіперплощинах.

Так в таблицях 1, 2, 3 наведено часові витрати на побудову автоматів для гіперплощин, що описані однією формулою АП:

Розмірність	2	2	2	8	8	8
Час виконання	0.0005	0.00061	0.00087	0.05221	0.06708	0.06529
Кількість станів	5	12	10	30	62	57

Таблиця 1

Розмірність	15	15	15	15	15	15
Час виконання	5.88305	8.14376	5.86654	5.94934	7.06429	7.72287
Кількість станів	80	86	77	78	77	78

Таблиця 2

Розмірність	15	15	15	15	15	15

Час виконання	12.31216	8.14376	5.86654	5.94934	7.06429	7.72287
Кількість станів	150	86	77	78	77	78

Таблиця 3

Таблиці 5, 6 демонструють витрати часу на класифікацію множини векторів, використовуючи вже побудовані автомати.

Розмірність	2	2	2	8	8	8
Середній час виконання для одного вектора	0.0001	0.00007	0.00004	0.00012	0.0002	0.00024
Кількість векторів	10	10	10	20	20	20

Таблиця 4

Розмірність	10	10	10	15	15	15
Середній час	0.00493	0.00051	0.00387	0.00647	0.00079	0.00011

виконання для одного вектора						
Кількість векторів	30	30	30	30	30	30

Таблиця 5

3.3 Опис інтерфейсу та можливостей

Реалізована система надає можливість класифікувати вектори відносно гіперплощини, заданої формулою арифметики Пресбургера. Для зручного користування системою створено зручний консольний інтерфейс. Нижче приведено список доступних команд та їх використання.

3.3.1 Доступні команди

Виклик допомоги по використанню—виводить список доступних команд:

```
python run.py -h
```

Запуск прикладів роботи:

```
python run.py -t
```

Побудова автомату для гіперплощини заданої як параметр:

```
python run.py -b <pa-formula>
```

Побудова автомату та класифікація векторів відносно заданої гіперплощини, дані задані як параметри:

```
python run.py -b <hyperplane> -v <vector-1>, ...,
<vector-n>
```

Побудова автомату та класифікація векторів відносно заданої гіперплощини, дані зчитані з вказаного файлу :

```
python run.py -f <filename>
```

Зберігання побудованого автомату у вказаний файл:

```
python run.py ... -s <filename>
```

3.3.2 Приклади роботи

Вивід доступних команд:

```
python run.py -h

Use commands described bellow:

            python run.py -t
                | run test example of work

            python run.py -b <pa-formula>
                | build automaton for given hyperplane
            python run.py -b <hyperplane> -v <vector-1>, ..., <vector-n>
                | build automaton and classify given vectors
```

```
python run.py -f <filename>
    | build automaton and classify vectors taken from file

python run.py ... -s <filename>
    | save built automaton to file
```

Запуск з випадово генерованими даними:

```
python run.py -t

Build for 0.09281 miliseconds
Automaton successfully built with 98 states
98

For hyperplane [18, 10, 0, -20, 2, 15, -18, -4, 11] = 31
```

```
Vector  [10, 17, 9, 8, 5, 0, 6, 1, 11]
Classify for 4e-05 milliseconds
UNDER hyperplane

Vector  [7, 6, 4, 16, 17, 10, 19, 15, 13]
Classify for 3e-05 milliseconds
BELOW hyperplane

Vector  [18, 17, 8, 4, 2, 3, 10, 0, 13]
Classify for 4e-05 milliseconds
UNDER hyperplane
```

Запуск побудови автомата для заданої формули:

```
python run.py -b 2x-4y+6z=12

Build for 0.00023 miliseconds
Automaton successfully built with 18 states
```

Запуск побудови та класифікації з заданими даними:

```
python run.py -b 2x-4y+6z=12 -v 2,3,4 7,2,1

Build for 0.00022 miliseconds
Automaton successfully built with 18 states
Classify for 2e-05 milliseconds
UNDER hyperplane
Classify for 1e-05 milliseconds
ON hyperplane
```

Побудова та класифікація з побальшим збереженням автомату:

```
python run.py -b 6x-12y+10z-2u=17 -v 2,34,14,10 8,2,4,25 -s

Build for 0.00074 miliseconds
Automaton successfully built with 33 states
Classify for 2e-05 milliseconds
BELOW hyperplane
Classify for 1e-05 milliseconds
ON hyperplane
Automaton saved to result-20210429-093700.txt
```

ВИСНОВКИ

В рамках даної роботи було досліджено ефективність застосування скінченних автоматів для вирішення задачі класифікації векторів відносно гіперплощини. Були реалізовані алгоритми побудови автоматів для формул Арифметики Пресбургера, так як формули АП і були використані у якості способу представлення гіперплощин. Код реалізації приведений у додатках.

Дана робота передбачає, що в такому описі буде міститись лише одна формула, проте наступним кроком стане допрацювання системи для охоплення випадків, коли для опису гіперплощини буде використовуватися система формул АП.

Сумарно було проведено близько 2000 вимірів швидкодії (по 100 вимірів для 10 випадків розімрностей для побудови автоматів та по 100 вимірів процесу класифікації векторів на кожній з обраних розмірностей). Результати частково були відображені у вигляді таблиць з середніми значеннями на кожні 25 вимірів.

Результати дають розуміння того, що процес побудови автоматів потребує певного часу, проте подальші кроки класифікації демонструють виправданість попередніх часових витрат.

Загалом, можна зробити висновок, що використання скінченних автоматів як інструменту для вирішення задачі класифікації векторів відносно гіперплощин є ефективним.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. *Кривий С.Л.* Лінійні діофантові обмеження та їх застосування. *Чернівці; Київ: Букрек, 2015. 224 с.*
2. *Anderson J.* Automata Theory with Modern Applications. *Cambridge University Press. UK, 2006. 255 p.*
3. *Büchi J. R.* Weak second-order arithmetic and finite automata. // *Z. Math Logic und Grundl. Math. 1960 №6 p. 66-92*
4. *Church A.* Logic, arithmetic, automata. // *In Proc. Intern. Mathem. Congr. 1962 p.21-32*
5. *Elgot C.C.* Decision problems of finite automata design and related arithmetics. // *Trans. Amer. Math. Soc. 1961. v. 98 p.21-52*
6. *Хопкрофт, Джон, Э., Мотвани, Раджив, Ульман, Джеффри, Д.* Введение в теорию автоматов, языков и вычислений, 2-е изд.. : *Пер. с англ. — М. : Издательский дом "Вильямс", 2008. — 528 с. : ил. — Парал. тит. англ. ISBN 978-5-8459-1347-0 (рус.)*
7. *Palagin A.V.,Opanasenko V.N., Kryvyi S.L.* Resource and Enerdy Optimisation Oriented Development of FPGA-Based Adaptive Logical Networks for Classification Problem. *In book: "Green IT Engineering: Components, Networks and Systems Implementation". Ed. Kharchenko V., Kondratenko Y., Kacprzyk J.– Springer-Verlag Berlin Htidel-berg. – 2016. –P. 195–218*
8. *Крывый С.Л., Опанасенко В.М.,* Разделение множества векторов с целыми неотрицательными координатами с использованием логических аппаратных средств. *ж. «Киберн. и сист. анализ». – 2018. – том 54. – № 2.– С. 150-159*
9. *Крывый С.Л., Опанасенко В.М., Завьялов С.Б.* Разбиение множества векторов с целыми координатами логическими аппаратными

средствами. *ж. «Кибернетика и системный анализ». – 2019. Том 55.-№ 3. - С.136–148*

10. *Крывый С.Л., Опанасенко В.М., Завьялов С.Б.* Логические операции над нечеткими множествами и отношениями в автоматной интерпретации. *ж. «Кибернетика и системный анализ». – 2020. т. 56.-№ 6. - С.175-183*
11. *Кривий С.Л.* Скінченні автомати: теорія, алгоритми, складність. *Чернівці; Київ: Букрек, 2020. 427 с.*

ДОДАТКИ

classification.py

```
from math import floor
import time


class Hyperplane:
    def __init__(self, coefs=list(), const_term=int(),
equation=""):
        if not (coefs or const_term or equation):
            raise Exception("Hyperplane is empty")
        if isinstance(coefs, str):
            equation = coefs
        if not equation:
            self.coefs = coefs
            self.const_term = const_term
        else:
            self.coefs, self.const_term =
self.str_to_equation(equation)
        self.dimension = len(self.coefs)

    def __str__(self):
        return str(self.coefs) + " = " +
str(self.const_term)

    def __repr__(self):
```

```
        return self.__str__()

    def str_to_equation(self, equation):
        from string import punctuation, ascii_letters
        equation = Vector.del_chars(equation, punctuation
+ ascii_letters, ".-+=")
        coefs, const_term = equation.split("=")
        to_int = lambda x: int(x) if x not in "+-" else
int(x + "1")
        try:
            const_term = to_int(const_term)
        except ValueError:
            raise Exception("Const term must be int type")
        coefs = map(to_int, coefs.split())
        return tuple(coefs), const_term


class Vector:
    def __init__(self, vector=""):
        if isinstance(vector, str):
            vector = self.str_to_vector(vector)
        self.vector = vector

    def __str__(self):
        return str(self.vector)

    def __repr__(self):
```

```
        return self.__str__()

    def str_to_vector(self, vector: str):
        from string import punctuation
        vector = self.del_chars(vector, punctuation, "-.")
        to_int = lambda x: int(x)
        vector = map(to_int, vector.split())
        return tuple(vector)

    @staticmethod
    def del_chars(vector: str, to_delete: str, exc=""):
        for elem in exc:
            to_delete = to_delete.replace(elem, "")
        for elem in to_delete:
            vector = vector.replace(elem, " ")
        return vector


class Automaton:
    def __init__(self, hyperplane: Hyperplane):
        try:
            self.__build(hyperplane)
        except Exception as e:
            self.__built = False
            print("Can't build automaton for input
hyperplane")
            print("Error:", e)
```

```
        else:
            self.__built = True
            print("Automaton successfully built with {}
states".format(len(self.states)))

    def __str__(self):
        if not self.__built:
            return "You hasn't build automaton"
        string = "\nAutomaton:\n\nStates:\n\t{}\n Initial
states:\n\t{}\n Final states:\n\t{}\n
Transitions:\n".format(
            self.states, self.init_state,
self.final_states)
        for state, transition in self.transitions.items():
            string += "  from {}:\n".format(state)
            for word, new_state in transition.items():
                string += "\tby {} -> {}\n".format(word,
new_state)
        return string

    def __repr__(self):
        return self.__str__()

    def __build(self, hyperplane: Hyperplane):
        timer = time.perf_counter()
        if not hyperplane.coefs:
            raise Exception("Hyperplane is empty!")
```

```
        b = hyperplane.const_term
        a = hyperplane.coefs
        n = hyperplane.dimension

        words = self.__get_all_words(n)
        workset = {b}
        a0 = b
        states = set()
        transitions = dict()
        final_states = set()

        while workset:
            s = workset.pop()
            states.add(s)
            if s >= 0:
                final_states.add(s)
            for z in words:
                j = floor((s - sum([el[0] * el[1] for el
in zip(a, z)])) / 2)
                if j not in states:
                    workset.add(j)
                try:
                    transitions[s][z] = j
                except KeyError:
                    transitions[s] = dict()
                    transitions[s][z] = j
```

```
        timer = float('%.5f' % (time.perf_counter() -
timer))
        print("Build for {} miliseconds".format(timer))
        self.states = states
        self.init_state = a0
        self.final_states = final_states
        self.transitions = transitions
        self.dimension = n

    def save(self, file):
        with open(file, "w") as f:
            f.write(self.__str__())
        print("Automaton saved to {}".format(file))

    def __get_all_words(self, dimension):
        n = dimension
        # make_bin_word = lambda i: tuple(int(xi) for xi
in (bin(i)[2:].zfill(n)))
        to_int = lambda x: int(x)
        make_bin_word = lambda i: tuple(map(to_int,
(bin(i)[2:].zfill(n))))
        words = list(map(make_bin_word, range(2 ** n)))
        return words

    def classify(self, vector: Vector):
        if not self.__built:
            raise Exception("You haven't build an
```

```
automaton")

        timer = time.perf_counter()
        vector = vector.vector
        n = len(bin(int(max(vector)))) - 2
        # n = self.dimension

        make_bin_word = lambda i:
reversed(tuple(map(lambda x: int(x),
(bin(i)[2:].zfill(n)))))
        vector = map(make_bin_word, vector)
        words = zip(*vector)

        state = self.init_state

        for word in words:
            # print("from {} by {} to {}".format(state,
word, self.transitions[state][word]))
            state = self.transitions[state][word]

        timer = float('%.5f' % (time.perf_counter() -
timer))
        print("Classify for {}
milliseconds".format(timer))
        if state == 0:
            print("ON hyperplane")
            return 0
```

```
        elif state > 0:
            print("BELOW hyperplane")
            return 1
        else:
            print("UNDER hyperplane")
            return -1


if __name__ == "__main__":
    k = -1
    a = (2, -1)
    x1 = (2, 5)
    x2 = (3, 7)
    x3 = (3, 5)

    v1 = Vector(vector=x1)
    v2 = Vector(vector=x2)
    v3 = Vector(vector=x3)

    v = [v1, v2, v3]

    hp = Hyperplane(a, k)
    s1 = "2x-y=0"
    s2 = "-4y+5=4"
    h1 = Hyperplane(s1)
    h2 = Hyperplane(s2)
    a = Automaton(hp)
```

```
    h3 = Hyperplane("4x+3y=24")
    v1 = Vector("6,0")
    v2 = Vector("2,4")
    v3 = Vector("4,4")
    a3 = Automaton(h3)

    for vi in v:
        print(a.classify(vi))
```

test.py

```
from classification import *
from random import randrange, sample


def test_from_file(file):
  with open(file, "r") as f:
     equation = f.readline()
     h = Hyperplane(equation)
     a = Automaton(h)
     print("\nFor hyperplane", h, end="\n")
     for line in f.readlines():
        vector = Vector(line)
        print("\nVector ", vector)
        a.classify(vector)
```

```
  return a

def test_auto_gen(file="", max_dim=10, max_test_cases=5):
  if max_dim == 10:
     dim = randrange(max_dim + 1)
  else:
     dim = max_dim
  test_cases = randrange(3, max_test_cases)

  coefs = sample(range(-20, 20), dim)
  const_term = randrange(-200, 200)
  vectors = list()

  h = Hyperplane(coefs, const_term)
  a = Automaton(h)
  print(len(a.states))
  print("\nFor hyperplane", h, end="\n")

  for i in range(test_cases):
     vector = sample(range(20), dim)
     vector = Vector(vector)
     vectors.append(vector)
     print("\nVector ", vector)
     a.classify(vector)

  if file:
```

```
    with open(file, "w") as f:
        f.writeline(h)
        f.writelines(vectors)
    print("Test case saved to ", file)

if __name__ == "__main__":
  a = test_from_file("test3.txt")
  #test_auto_gen(max_dim=15)
```

run.py

```
import sys
import time
from classification import *
from test import *


if len(sys.argv) > 1:
   automaton = None
   # noinspection PyBroadException
   print("python", end=" ")
   print(*sys.argv, end="\n\n")
   try:
       if "-h" in sys.argv[1:]:
           print("""Use commands described bellow:\n
           python run.py -t
               | run test example of work\n
```

```
            python run.py -b <pa-formula>
                | build automaton for given hyperplane
            python run.py -b <hyperplane> -v <vector-1>,
..., <vector-n>
                | build automaton and classify given
vectors\n
            python run.py -f <filename>
                | build automaton and classify vectors
taken from file\n
            python run.py ... -s <filename>
                | save built automaton to file\n
             """)
        elif sys.argv[1] == "-t":
            test_auto_gen()
        elif len(sys.argv) == 3 and sys.argv[1] == "-b":
            pa_formula = sys.argv[2]
            automaton =
Automaton(Hyperplane(equation=pa_formula))
        elif len(sys.argv) > 3 and sys.argv[1] == "-b" and
sys.argv[3] == "-v":
            pa_formula = sys.argv[2]
            automaton =
Automaton(Hyperplane(equation=pa_formula))
            for vector in sys.argv[4:]:
                if vector != "-s":
                    automaton.classify(Vector(vector))
        elif len(sys.argv) == 3 and sys.argv[1] == "-f":
```

```
            file = sys.argv[2]
            automaton = test_from_file(file)
        if "-s" in sys.argv[1:] and "-t" not in
sys.argv[1:]:
            file =
"result-{}.txt".format(time.strftime("%Y%m%d-%H%M%S"))
            automaton.save(file)
    except Exception as e:
        print("Wrong number of arguments\n")
        print(e)
```