Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

Кафедра безпеки інформаційних систем і технологій

Дипломна робота

*На тему*

«Генерація криптографічних булевих функцій для нелінійних вузлів ускладнення симетричних шифрів із застосуванням методів імітації відпалу»

Виконав:

Студент групи КБ-41

Кравченко Є.М.

Перевірив:

Харків – 2020 р.

**РЕФЕРАТ**

**ЗМІСТ**

# **ВСТУП**

# **Аналіз відомих методів генерації нелінійних вузлів ускладнення симетричних шифрів**

# **Обґрунтування критеріїв та показників ефективності нелінійних вузлів ускладнення симетричних шифрів**

# **Обґрунтування вибору напрямку дослідження**

# **Дослідження методу та обчислювальних алгоритмів імітації відпалу**

# **Застосування методу імітації відпалу для пошуку криптографічних булевих функцій**

# **ОБҐРУНТУВАННЯ ВИБОРУ МОВИ ПРОГРАМУВАННЯ**

Мова програмування (англ. Programming language) — це штучна мова, створена для передачі команд машинам, зокрема комп'ютерам. Мови програмування використовуються для створення програм, котрі контролюють поведінку машин, та запису алгоритмів.

Більш строге визначення: мова програмування — це система позначень для опису алгоритмів та структур даних[1], певна штучна формальна система, засобами якої можна виражати алгоритми[2]. Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням.[джерело?]

З часу створення перших програмованих машин було створено понад дві з половиною тисячі мов програмування.[3] Щороку їх кількість поповнюється новими. Деякими мовами вміє користуватись тільки невелике число їх власних розробників, інші стають відомі мільйонам людей. Професійні програмісти зазвичай застосовують в своїй роботі декілька мов програмування.

## Класифікація мов програмування

Мови класифікують за такими критеріями[джерело?]:

Рівень абстракції

Мови програмування високого рівня оперують сутностями ближчими людині, такими як об'єкти, змінні, функції. Мови програмування нижчого рівня оперують сутностями ближчими машині: байти, адреси, інструкції. Текст програми на мові високого рівня зазвичай набагато коротший ніж текст такої самої програми на мові низького рівня, проте програма має більший розмір.

Область застосування

Універсальні та спеціалізовані. Спеціалізовані мови теж бувають Тьюрінг-повні, та все ж їх область застосування обмежена, як наприклад у мови shell.

Підтримувані парадигми програмування

Об'єктно-орієнтовані, логічні, функційні, структурні…

Імперативні мови базуються на ідеї змінної, значення якої змінюється присвоєнням. Вони називаються імперативними (лат. imperative — наказовий), оскільки складаються із послідовностей команд, які звичайно містять присвоєння змінних де вираз може посилатися на значення змінних присвоєних попередніми командами.

Способи реалізації мов

Мови програмування можуть бути реалізовані як компільовані та інтерпретовані.

Програма на компільованій мові за допомогою компілятора (особливої ​​програми) (компілюється) в машинний код (набір інструкцій) для даного типу процесора, що записується у об'єктний модуль. З одного або кількох об'єктних файлів компонувальник формує виконуваний файл, який може бути запущений на виконання як окрема програма. Іншими словами, компілятор переводить вихідний текст програми з мови програмування високого рівня в двійкові коди інструкцій процесора.

Якщо програма написана на скриптовій мові, то інтерпретатор безпосередньо виконує (інтерпретує) вихідний текст без попереднього перекладу. При цьому програма залишається мовою оригіналу і не може бути запущена без інтерпретатора. Процесор комп'ютера, в зв'язку з цим, можна назвати інтерпретатором для машинного коду.

Поділ на компільовані і інтерпретовані мови є умовним. Так, для будь-якої традиційно компілючої мови, як, наприклад, Паскаль, можна написати інтерпретатор. Крім того, більшість сучасних «чистих» інтерпретаторів не виконують конструкції мови безпосередньо, а компілюють їх в деяке високорівневе проміжне представлення (наприклад, з розіменуванням змінних і розкриттям макросів).

Для будь-якої інтерпритуючої мови можна створити компілятор — наприклад, мова Лісп, початково інтерпретована, може компілюватися без обмежень. Створюваний під час виконання програми код може так само динамічно компілюватися під час виконання.

Як правило, скомпільовані програми виконуються швидше і не вимагають для виконання додаткових програм, так як вже переведені на машинну мову. Разом з тим, при кожній зміні тексту програми потрібно її перекомпіляція, що уповільнює процес розробки. Крім того, скомпільована програма може виконуватися тільки на тому ж типі комп'ютерів і, як правило, під тією ж операційною системою, на яку був розрахований компілятор. Щоб створити виконуваний файл для машини іншого типу, потрібна нова компіляція.

Інтерпретовані мови володіють деякими специфічними додатковими можливостями (див. вище), крім того, програми на них можна запускати відразу ж після зміни, що полегшує розробку. Програма на скриптовій мові може бути найчастіше запущена на різних типах машин та операційних систем без додаткових зусиль.

Однак інтерпретовані програми виконуються помітно повільніше, ніж компільовані, крім того, вони не можуть виконуватися без програми-інтерпретатора.

Деякі мови, наприклад, Java та C #, перебувають між компільованими і інтерпретованими. А саме, програма компілюється не в машинну мову, а в машинно-незалежний код низького рівня, байт-код. Далі байт-код виконується віртуальною машиною. Для виконання байт-коду зазвичай використовується інтерпретація, хоча окремі його частини для прискорення роботи програми можуть бути трансльовані в машинний код безпосередньо під час виконання програми за технологією компіляції «на льоту» (Just-in-time compilation, JIT). Для Java байт-код виконується віртуальною машиною Java (Java Virtual Machine, JVM), для C # — Common Language Runtime.

Подібний підхід у деякому сенсі дозволяє використовувати плюси як інтерпретаторів, так і компіляторів. Слід згадати, що є мови, які мають і інтерпретатор, і компілятор (Форт (Forth)).

Процедури та функції

Підпрограми діляться на процедури та функції: Синтаксично процедури та функції складаються з заголовка (що містить ключове слово procedure або function, ім'ям, за яким може слідувати опис передаваних параметрів в дужках, тип повертаного значення через символ двокрапки для функцій і крапки з комою для процедур), після заголовка слідує 'тіло', після якого ставиться символ ;.

Тіло процедури, як і програми, у свою чергу може містити описи процедур і функцій. Таким чином, процедури і функції можуть бути вкладені один в одного як завгодно глибоко, при цьому тіло програми — саме верхнє в ланцюжку.

Причому вміст секцій опису змінних, типів, констант, зовнішнього тіла (процедури, функції, програми), розташованих перед описом процедури/функції, доступні усередині неї. Також, в більшості діалектів з процедури можна звертатися до параметрів зовнішньої процедури.

Услід за заголовком процедур/функцій замість тіла може поміщатися ключове слово forward, це робиться в тому випадку, якщо опис процедури/функції розташовується в програмі після її виклику, і пов'язано з підтримуваною в Паскалі можливістю компіляції програми за один прохід.

Процедури відрізняються від функцій тим, що функції повертають якесь значення, а процедури — ні.

Об'єктно-орієнтоване програмування

Докладніше: Об'єктно-орієнтована мова програмування та Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (ООП) — це технологія створення складного програмного забезпечення, яке засноване на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію із спадкоємством властивостей.

Основна перевага ООП — це значне спрощення процесів створення та модифікації програмних систем. Набагато легше маніпулювати 100 об'єктами, кожен з яких сам відповідає за свою поведінку і обробку даних пов'язаних з ним, ніж тисячами функцій розкиданих по різних модулях.

Основні недоліки в ООП — це деяке зниження швидкодії через складнішу організацію програмної системи, а також, як правило, помітне збільшення об'єму бінарного коду (особливо при використанні стандартних бібліотек класів в невеликих програмах) через те, що більшість сучасних компіляторів і компонувальників не здатні виявити і видалити весь код, що доводиться на невживані класи, віртуальні методи і інші елементи ООП.

Семантика мов програмування

Докладніше: Семантика мов програмування

Існує кілька підходів до визначення семантики мов програмування.

Найбільш широко поширені наступні три різновиди семантик: операційна, дериваційна (аксіоматична) і денотаційна (математична)[джерело?].

При описі семантики в рамках операційного підходу зазвичай виконання конструкцій мови програмування інтерпретується за допомогою деякої уявної (абстрактної) ЕОМ.

Дериваційна семантика описує наслідки виконання конструкцій мови за допомогою мови логіки і завдання перед- і пост-умов.

Денотаційна семантика оперує поняттями, типовими для математики — множини, відповідності, а також судження, твердження та ін.

## Мови програмування низького рівня

Низькорівнева мова програмування — мова програмування наближена до машинного коду. Перші комп'ютери доводилось програмувати двійковими машинними кодами. Проте програмувати таким чином — доволі трудомістке і важке завдання з тої причини, що людям важко запам'ятовувати цифрові позначення команд[1]. Для спрощення цього завдання почали з'являтися мови програмування низького рівня, які дозволяли задавати машинні команди в зрозумілішому для людини вигляді. Для перетворення їх у двійковий код були створені спеціальні програми — транслятори.[2]

Найбільш застосовуваними трансляторами є:

компілятори — перетворюють текст програми в машинний код, який можна зберегти і після цього використовувати уже без компілятора (прикладом є виконувальні файли з розширенням \*.exe).

інтерпретатори — перетворюють частину програми в машинний код, виконують і після цього переходять до наступної частини. При цьому щоразу при виконанні програми використовується інтерпретатор.

Прикладом мови низького рівня є Мова асемблера — мова (вірніше сімейство мов), оператори якої є мнемонічним позначенням машинних команд процесора. Деякі мови високого рівня (наприклад, C/C++) навіть дозволяють вставляти фрагменти асемблерного коду безпосередньо у тіло програми. Мови низького рівня орієнтовані на конкретний тип процесора і враховують його особливості, тому для перенесення асемблерної програми на іншу апаратну платформу її потрібно майже цілком переписати. Певні відмінності є і в синтаксисі програм під різні компілятори. Щоправда, центральні процесори для комп'ютерів фірм AMD та Intel практично сумісні і відрізняються лише деякими специфічними командами. А ось спеціалізовані процесори для інших пристроїв, наприклад, відеокарт, телефонів містять суттєві відмінності.

В даний час існують бібліотеки, які дозволяють писати на асемблері повноцінні програми з графічним інтерфейсом користувача[3].

Різноманітні байткоди, які виконуються віртуальними машинами (як то байткод мови програмування Java)також по суті є низькорівневими мовами, щоправда байткоди переважно не пишуться вручну, а транслюються з відповідної високорівневої мови. Згодом цей код інтерпретується віртуальною машиною. Команди байткода схожі на асемблерні. Перевагаою такого підходу є те, що програмісти можуть легко писати програми на високорівневій мові і після компіляції в байткод, віртуальна машина може швидше інтерпретувати байткод, який більш наближений до машинного коду ніж початковий код програми. Перевагою використання віртуальної машини і байткоду є деяке унезалежнення від апаратної частини. Недоліком байткоду є дещо нижча швидкодія програм, порівняно з програмами, з самого початку скомпільованими в машинний код.

Переваги

За допомогою мов низького рівня створюють ефективні і компактні програми, оскільки розробник отримує доступ до всіх можливостей процесора.

Мови низького рівня, як правило, використовують для написання невеликих системних програм, драйверів пристроїв, модулів стиків з нестандартним обладнанням, програмування спеціалізованих мікропроцесорів, коли найважливішими вимогами є компактність, швидкодія і можливість прямого доступу до апаратних ресурсів.

Недоліки

Програміст, що працює з мовами низького рівня, має бути високої кваліфікації, добре розуміти будову мікропроцесорної системи, для якої створює програму. Так, якщо програму створюють для комп'ютера, потрібно знати будову комп'ютера і, особливо, влаштування і особливості роботи його процесора.

результуюча програма не може бути легко перенесена на комп'ютер або пристрій з іншим типом процесора.

значний час розробки великих і складних програм.

# **ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ ІМІТАЦІЇ ВІДПАЛУ**

### **Обґрунтування рекомендацій щодо провадження отриманих результатів**

# **ВИСНОВКИ**

# **РЕКОМЕНДАЦІЇ**

# **ПЕРЕЛІК ПОСИЛАНЬ**