Доработка статического анализатора LuNA-программ prolog-analyzer

Выполнил:
студент гр. 22221 Мичуров Михаил Антонович

Руководитель ВКР:
Власенко Андрей Юрьевич, доцент каф. ПВ ФИТ

prolog-analyzer

- LuNA-программа представляется набором фактов на языке Prolog;
- Правила на Prolog используются для поиска ошибок.

Задачи

- *[Сделано]* Переработать способ обнаружения использования ФД в качестве входных в программе:
  - Учитывать использование ФД в границах циклов;
  - Учитывать только те использования значения ФД, где это значение запрашивается.
- *[Частично]* Добавить (ограниченную) поддержку условного оператора: (для условных выражений, не содержащих индексированных ФД)
  - *[Частично]* Проверка на тождественную истинность/ложность;
  - Проверка эквивалентности логических выражений;
  - Поиск ошибок с учетом возможных комбинаций истинности и ложности условий.

Поддержка условных операторов

Подзадачи:

- Поддержка индексированных имен;
- Проверка логических выражений на эквивалентность:

  Дает возможность:

  - обнаруживать тавтологии (выражение эквивалентно истине/лжи),
  - выполнять анализ с учетом того, что некоторые условия истинны или ложны одновременно.
- Приведение условий к виду, поддерживаемому CLP(B).

CLP(B): Constraint Logic Programming over Boolean Variables

CLP(B) - библиотека, реализующая парадигму программирования в ограничениях над булевыми значениями.

Markus Triska, Boolean constraints in SWI-Prolog: A comprehensive system description, Science of Computer Programming, Volume 164, 2018, Pages 98-115, ISSN 0167-6423.

Triska, M. (2016). The Boolean Constraint Solver of SWI-Prolog (System Description). In: Kiselyov, O., King, A. (eds) Functional and Logic Programming. FLOPS 2016. Lecture Notes in Computer Science(), vol 9613. Springer, Cham.

CLP(B): Constraint Logic Programming over Boolean Variables

Необходимые предикаты:

sat(+Expr)

Истина ⇔ *Expr* выполнимо.

taut(+Expr, -T)

Если *Expr* - тавтология, выполняется успешно с *T = 1*. Если *Expr* не выполнимо, выполняется успешно с *T = 0*. Иначе не выполняется.

Предикаты работают с формулами в формате, определенном библиотекой!

Получение формул из условий

Шаг 1. Избавиться от индексированных имен;

Шаг 2. Избавиться от арифметических выражений;

Шаг 3. Выразить сравнения через ‘==’, ‘<’ и отрицание;

Шаг 4. Заменить сравнения булевыми переменными.

Полученная в шаге 4 формула может быть конвертирована в формат CLP(B).

Пример получения формулы из условия

Изначальное выражение: `(x[N] >= y + 1 && x[N + y * 0] < 1 + y) && y > 5`

Пример получения формулы из условия

Изначальное выражение: `(x[N] >= y + 1 && x[N + y * 0] < 1 + y) && y > 5`

Шаг 1: **`(Ref_1 >= y + 1 && Ref_1 < 1 + y) && y > 5`**

`Ref_1` **=** `x[N]`

Пример получения формулы из условия

Изначальное выражение: `(x[N] >= y + 1 && x[N + y * 0] < 1 + y) && y > 5`

Шаг 1: `(Ref_1 >= y + 1 && Ref_1 < 1 + y) && y > 5`

`Ref_1` **=** `x[N]`

Шаг 2: **`(Arith_1 >= Arith_2) && (Arith_1 < Arith_2) && Arith_3 > Arith_4`**

`Arith_1` **=** `Ref_1`, `Arith_2 = y + 1`, `Arith_3` **=** `y`, `Arith_4` **=** `5`

Пример получения формулы из условия

Изначальное выражение: `(x[N] >= y + 1 && x[N + y * 0] < 1 + y) && y > 5`

Шаг 1: `(Ref_1 >= y + 1 && Ref_1 < 1 + y) && y > 5`

`Ref_1` **=** `x[N]`

Шаг 2: `(Arith_1 >= Arith_2) && (Arith_1 < Arith_2) && Arith_3 > Arith_4`

`Arith_1` **=** `Ref_1`, `Arith_2 = y + 1`, `Arith_3` **=** `y`, `Arith_4` **=** `5`

Шаг 3: **`!(Arith_1 < Arith_2) && (Arith_1 < Arith_2) && Arith_4 < Arith_3`**

Пример получения формулы из условия

Изначальное выражение: `(x[N] >= y + 1 && x[N + y * 0] < 1 + y) && y > 5`

Шаг 1: `(Ref_1 >= y + 1 && Ref_1 < 1 + y) && y > 5`

`Ref_1` **=** `x[N]`

Шаг 2: `(Arith_1 >= Arith_2) && (Arith_1 < Arith_2) && Arith_3 > Arith_4`

`Arith_1` **=** `Ref_1`, `Arith_2 = y + 1`, `Arith_3` **=** `y`, `Arith_4` **=** `5`

Шаг 3: `!(Arith_1 < Arith_2) && (Arith_1 < Arith_2) && Arith_4 < Arith_3`

Шаг 4: **`!Cond_1 && Cond_1 && Cond_2`**

`Cond_1` **=** `(Arith_1 < Arith_2)`, `Cond_2` **=** `(Arith_4 < Arith_3)`

Ошибки, выделенные в отдельные классы

- Формула в условии является тождественно ложной/истинной (LUNA23);
  - x > y || x <= y
- Условие выполняется всегда (не выполняется никогда) на всех последовательностях исполнения (LUNA24);
- Булево значение использовано в числовом контексте (LUNA25).
  - x > y > z

Тождественно истинная/ложная формула в условии

В LuNA-программе:

```
34      if (x[N] >= 0 && x[N + y * 0] < 0) && y > 5 {
35          print(x[N]);
36      }
```

Обнаруженная ошибка:

```
In: ./main.fa:34:if
Condition (((LUNA_x[LUNA_N] >= 0) && (LUNA_x[(LUNA_N + (LUNA_y * 0))] < 0)) && (LUNA_y > 5)) is always false
```

Использование булева значения как числа

В LuNA-программе:

```
26      if 0 < x < N {
27          print(x);
28      }
```

Обнаруженная ошибка:

```
In: ./main.fa:26:if
Boolean value used in numerical context: argument 0 of ((0 < LUNA_x) < LUNA_N)
```

Дальнейшая работа

- Анализ с учетом комбинаций истинности условий;
- Реализация обнаружения LUNA24 (условие одинаково на всех последовательностях исполнения).

Спасибо за внимание!

Булево выражение в CLP(B)

A *Boolean expression* is one of:

`0`	false
`1`	true
variable	unknown truth value
atom	universally quantified variable
~ *Expr*	logical NOT
Expr + *Expr*	logical OR
Expr * *Expr*	logical AND
Expr # *Expr*	exclusive OR
Var ^ *Expr*	existential quantification
Expr =:= *Expr*	equality
Expr =\= *Expr*	disequality (same as #)
Expr =< *Expr*	less or equal (implication)
Expr >= *Expr*	greater or equal
Expr < *Expr*	less than
Expr > *Expr*	greater than
`card(Is,Exprs)`	cardinality constraint (*see below*)
`+(Exprs)`	n-fold disjunction (*see below*)
`*(Exprs)`	n-fold conjunction (*see below*)

where *Expr* again denotes a Boolean expression.

Процесс преобразования условия в Prolog

?- Cond = ["&&", ["&&", [">=", luna_ref(["LUNA_x", "LUNA_N"]), 0], ["<", luna_ref(["LUNA_x", ["+", "LUNA_N", ["*", "LUNA_y", 0]]]), 0]], [">", "LUNA_y", 5]], condition_clpb(Cond, CondWithoutRefs, CondWithoutArith, CondNormalized, Formula, ClpbTermStr, ClpbTerm).

Cond = ["&&", ["&&", [">=", luna_ref(["LUNA_x", "LUNA_N"]), 0], ["<", luna_ref(["LUNA_x"|...]), 0]], [">", "LUNA_y", 5]],

CondWithoutRefs = ["&&", ["&&", [">=", "Ref_1", 0], ["<", "Ref_1", 0]], [">", "LUNA_y", 5]],

CondWithoutArith = ["&&", ["&&", [">=", "Arith_1", "Arith_2"], ["<", "Arith_1", "Arith_2"]], [">", "Arith_3", "Arith_4"]],

CondNormalized = ["&&", ["&&", ["!", ["<", "Arith_1", "Arith_2"]], ["<", "Arith_1", "Arith_2"]], ["<", "Arith_4", "Arith_3"]],

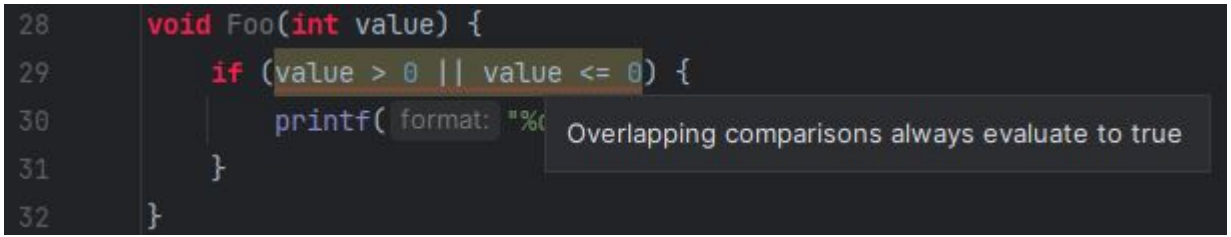
Formula = ["&&", ["&&", ["!", "Cond_1"], "Cond_1"], "Cond_2"],

ClpbTermStr = "(((~ Cond_1) * Cond_1) * Cond_2)",

ClpbTerm = ~_A*_A*_.

Пример правил для переписывания выражения

```
condition_formula(Cond, AliasesIn, AliasesOut, Formula) :-
    Cond = [Op, _, _],
    comparison_operator(Op),
    get_dict(CondKey, AliasesIn, Cond),
    AliasesOut = AliasesIn,
    cond_make_alias(CondKey, Formula),
    !.
condition_formula(Cond, AliasesIn, AliasesOut, Formula) :-
    Cond = [Op, _, _],
    comparison_operator(Op),
    dict_max_key(AliasesIn, MaxKey),
    CondKey is MaxKey + 1,
    put_dict([CondKey=Cond], AliasesIn, AliasesOut),
    cond_make_alias(CondKey, Formula),
    !.
condition_formula([Op, Arg], AliasesIn, AliasesOut, Formula) :-
    logic_operator(Op),
    condition_formula(Arg, AliasesIn, AliasesOut, ArgFormula),
    Formula = [Op, ArgFormula],
    !.
```


28 void Foo(int value) {
29 if (value > 0 || value <= 0) {
30 printf(format: "%
31 }
32 }
Overlapping comparisons always evaluate to true