Uczenie maszynowe i formalna weryfikacja dla pozyskiwania wiedzy w heurystycznej syntezie programów

Iwo Błądek

Wydział Informatyki i Telekomunikacji, Politechnika Poznańska

20.05.2022

Promotor:
prof. dr hab. inż. Krzysztof Krawiec (Politechnika Poznańska)

Recenzenci:
dr hab. Wojciech Jamroga, prof. IPI PAN (Université du Luxembourg; IPI PAN)
prof. dr hab. inż. Robert Schaefer (Akademia Górniczo-Hutnicza im. Stanisława Staszica)

Cele rozprawy doktorskiej

- Opracowanie metod wzbogacania programowania genetycznego dodatkową informacją w celu poprawy jego skuteczności i szybkości działania dla problemów syntezy programów.
- Stworzenie ewolucyjnych algorytmów syntezy programów z gwarancjami poprawności.
- Analiza zdolności uogólniania w przypadku specyfikacji formalnej określonej zbiorem ograniczeń logicznych.
- Eksperymentalna ocena algorytmów na zbiorach benchmarków znanych z literatury, a tam gdzie ich brak utworzenie własnych benchmarków na podstawie problemów zaczerpniętych z dziedzin m.in. programowania, matematyki i fizyki.

Plan prezentacji

Synteza programów komputerowych

Problem syntezy programów*

Problem syntezy programów zdefiniowany jest parą $(\mathcal{P}, Correct)$, gdzie $\mathcal{P}$ jest zbiorem dopuszczalnych programów (*język programowania*), a *Correct* jest funkcją $\mathcal{P} \rightarrow \{0, 1\}$ nazywaną *predykatem poprawności*. Rozwiązanie problemu syntezy programów $(\mathcal{P}, Correct)$ polega na znalezieniu takiego $p \in \mathcal{P}$, że $Correct(p) = 1$.

* Źródło: Krzysztof Krawiec, "Behavioral Program Synthesis with Genetic Programming", 2016.

Przykład nr 1:

- Automatyczne transformacje danych w arkuszach kalkulacyjnych.

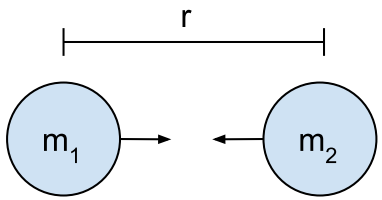
	A	B
1	**Name**	**Properly formatted name**
2	ronnie anderson	Anderson, Ronnie
3	tom boone	Boone, Tom
4	sally brook	Brook, Sally
5	jeremy hill	Hill, Jeremy
6	mattias waldau	Waldau, Mattias
7	robert furlan	Furlan, Robert
8	david white	White, David

	A	B	C	D
1	**Last name**	**First name**	**Domain**	**Email address**
2	Anderson	Ronnie	Gmail	r.anderson@gmail.com
3	Boone	Tom	Hotmail	t.boone@hotmail.com
4	Brook	Sally	Outlook	s.brook@outlook.com
5	Hill	Jeremy	Gmail	j.hill@gmail.com
6	Waldau	Mattias	Hotmail	m.waldau@hotmail.com
7	Furlan	Robert	Outlook	r.furlan@outlook.com
8	White	David	Gmail	d.white@gmail.com

$\mathcal{P}$: język formuł arkusza kalkulacyjnego
Correct: zgodność z przykładami użytkownika

Przykład nr 2:

- Regresja symboliczna z ograniczeniami.



Przykłady:

m_1	m_2	r	oczekiwany wynik
10.88386	11.36099	15.74871	0.000000000033273260096895905
11.1782	3.21214	7.67932	0.000000000040635631498539907
...			

Ograniczenia:

- $f(m_1, m_2, r) = f(m_2, m_1, r)$
- $f(m_1, m_2, r) \geq 0$
- silna monotoniczność f ze względu na m_1 i m_2

$\mathcal{P}$: wyrażenia matematyczne
Correct: minimalny błąd na przykładach i spełnienie ograniczeń

Specyfikacja programu

Informacja podawana przez użytkownika w celu zdefiniowania poprawnego działania programu. Jest podstawą do utworzenia predykatu poprawności *Correct*.

- **Przykłady wejście-wyjście**

x	y	max(x,y)
0	0	0
1	0	1
3	4	4
...	...	...

- **Formalna specyfikacja**

$$\forall_{x,y} \quad max(x,y) \geq x \quad \wedge$$
$$max(x,y) \geq y \quad \wedge$$
$$(max(x,y) = x \ \vee \ max(x,y) = y)$$

Specyfikacja programu

Specyfikacja programu

Informacja podawana przez użytkownika w celu zdefiniowania poprawnego działania programu. Jest podstawą do utworzenia predykatu poprawności *Correct*.

- **Demonstracja kroków wykonania**
- **Opis w języku naturalnym**
- **Częściowe lub nieoptymalne programy**

Klasa rozpatrywanych programów

W rozprawie rozpatrujemy syntezę programów:

- **deterministycznych**
- **funkcyjnych**, tzn. rezultat wykonania dowolnej instrukcji programu jest w pełni reprezentowany przez zwracaną przez nią wartość wyjściową
- **bez pętli i wywołań rekurencyjnych**
- **spełniających warunek stopu**

Ponadto zakładamy również, że dla każdego rozpatrywanego zadania syntezy zawsze istnieje program $p \in \mathcal{P}$ taki że $\mathit{Correct}(p) = 1$.

Plan prezentacji

Formalna weryfikacja

Celem formalnej weryfikacji jest zagwarantowanie, że program posiada pewne własności dla wszystkich możliwych wejść.

Realizowane jest to poprzez udowodnienie dla programu p, że:

$$\forall_{in}\ Pre(in) \implies Post(in, p(in)), \tag{1}$$

gdzie $p(in)$ to wynik działania p dla wejścia in, Pre jest **warunkiem wstępnym**, a $Post$ jest **warunkiem końcowym**.

Alternatywne sformułowanie zadania:

$$\exists_{in}\ Pre(in) \implies \neg Post(in, p(in)). \tag{2}$$

Spełnialność Modulo Teorie (SMT)

Problem decyzyjny polegający na stwierdzeniu, czy dana formuła w logice pierwszego rzędu jest spełnialna dla danej teorii τ, która definiuje semantykę występujących w niej symboli.

Przykłady:

$a, b \in \{false, true\}$

$\neg a \ \vee \ b$
SAT: $a = false, \ b = true$

$a \ \wedge \ \neg a \ \wedge \ b$
UNSAT

$x, y, z \in \mathbb{Z}$
$a \in \{false, true\}$

$(10 \cdot x = 20) \ \wedge \ a$
SAT: $x = 2, \ a = true$

$(x < y) \ \wedge \ (y < z) \ \wedge \ (z < x)$
UNSAT

Spełnialność Modulo Teorie (SMT)

Problem decyzyjny polegający na stwierdzeniu, czy dana formuła w logice pierwszego rzędu jest spełnialna dla danej teorii τ, która definiuje semantykę występujących w niej symboli.

Przykłady:

$x, y \in \mathbb{Z}$

$x^2 + 1 \leq 2 \cdot x$
SAT: $x = 1$

$\forall_{x,y}\ (x + y)^2 > x^2 + y^2$
UNSAT

$x, y \in \texttt{String}$

$\texttt{str.substr}(x, 0, 2) = \texttt{str.substr}(x, 2, 2)\ \wedge$
$\texttt{str.at}(x, 0) \neq \texttt{str.at}(x, 1)$
SAT: $x =$ `"abab"`

$\texttt{str.len}(x \texttt{ ++ } y) \neq \texttt{str.len}(x) + \texttt{str.len}(y)$
UNSAT

Przykład formalnej weryfikacji przy użyciu SMT

Niepoprawny program MAX:

```
if x < y:
    return x
else:
    return y
```

Formalna specyfikacja:

$$\begin{aligned} max(x,y) \geq x \quad \wedge \\ max(x,y) \geq y \quad \wedge \\ (max(x,y) = x \ \vee \ max(x,y) = y) \end{aligned}$$

Wynik zwrócony przez solwer SMT:

```
SAT
x = -1
y = 0
```

SAT oznacza, że program jest niepoprawny, a solwer zwraca **kontrprzykład** $(-1, 0)$.

Przykład formalnej weryfikacji przy użyciu SMT

Poprawny program MAX:

```
if x > y:
    return x
else:
    return y
```

Formalna specyfikacja:

$$\begin{aligned} max(x, y) \geq x \quad \wedge \\ max(x, y) \geq y \quad \wedge \\ (max(x, y) = x \ \vee \ max(x, y) = y) \end{aligned}$$

Wynik zwrócony przez solwer SMT:

```
UNSAT
```

UNSAT oznacza, że program jest poprawny ze względu na specyfikację. Kontrprzykład nie istnieje.

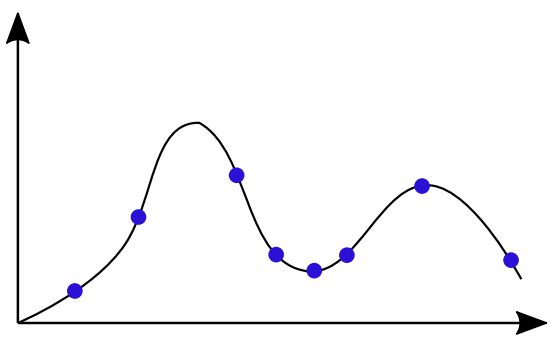
Plan prezentacji

Programowanie genetyczne

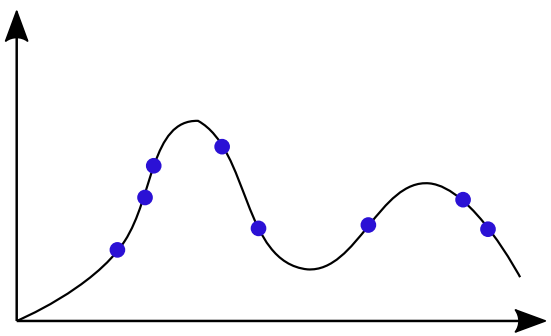
Najważniejsze cechy programowania genetycznego (GP):

- Heurystyczny algorytm optymalizacji globalnej.
- Na każdym etapie utrzymywana jest populacja rozwiązań kandydackich.
- Rozwiązania kandydackie są premiowane za bliskość do celu optymalizacji i jest większa szansa, że zostaną wybrane do reprodukcji
- Reprodukcja wprowadza drobne zmiany w rozwiązaniach i/lub wymienia ich fragmenty.

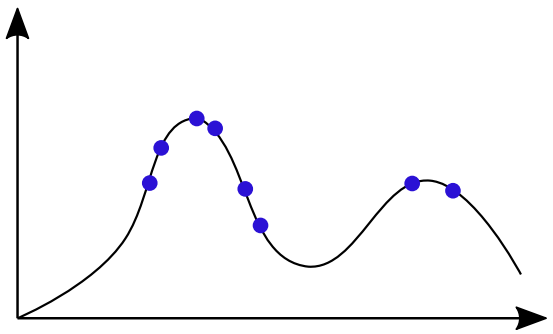
Ilustracja działania algorytmu w czasie:



Pokolenie nr 1



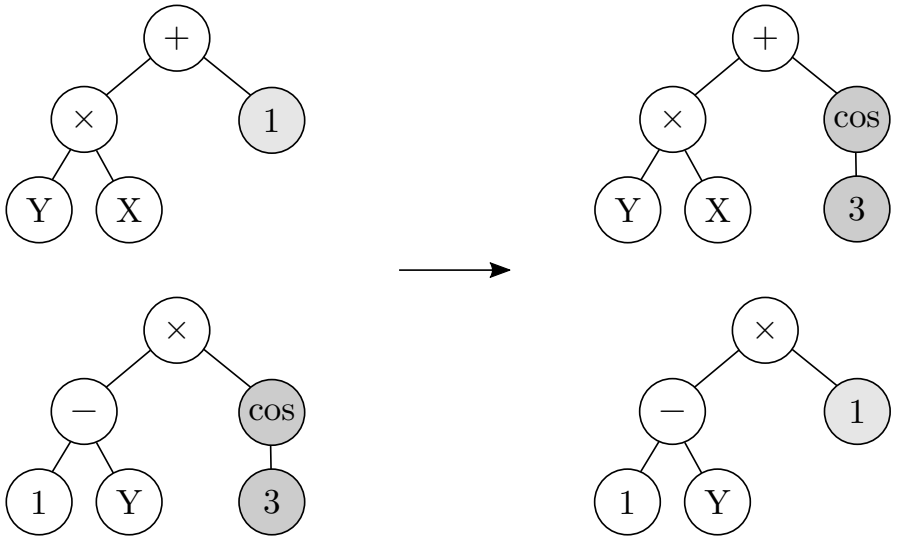
Pokolenie nr 10



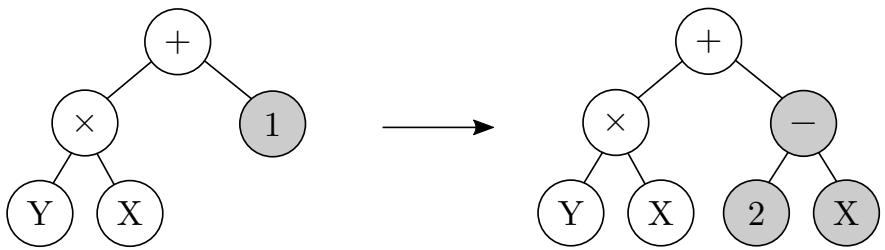
Pokolenie nr 20

Reprezentacja programów i operatory wariacji

- Programy reprezentowane są w postaci drzew instrukcji.
- **Krzyżowanie** tworzy nowego osobnika łącząc cechy istniejących.



- **Mutacja** dokonuje drobnej modyfikacji w istniejącym rozwiązaniu.



Plan prezentacji

Ewolucyjne szkicowanie programów (EPS)

Problem badawczy:

- W paradygmecie *syntezy programów przez szkicowanie*[1] oprócz specyfikacji w postaci testów użytkownik musi dostarczyć szkic rozwiązania zawierający luki, które mają zostać uzupełnione.
- Czy da się zautomatyzować proces tworzenia szkiców?

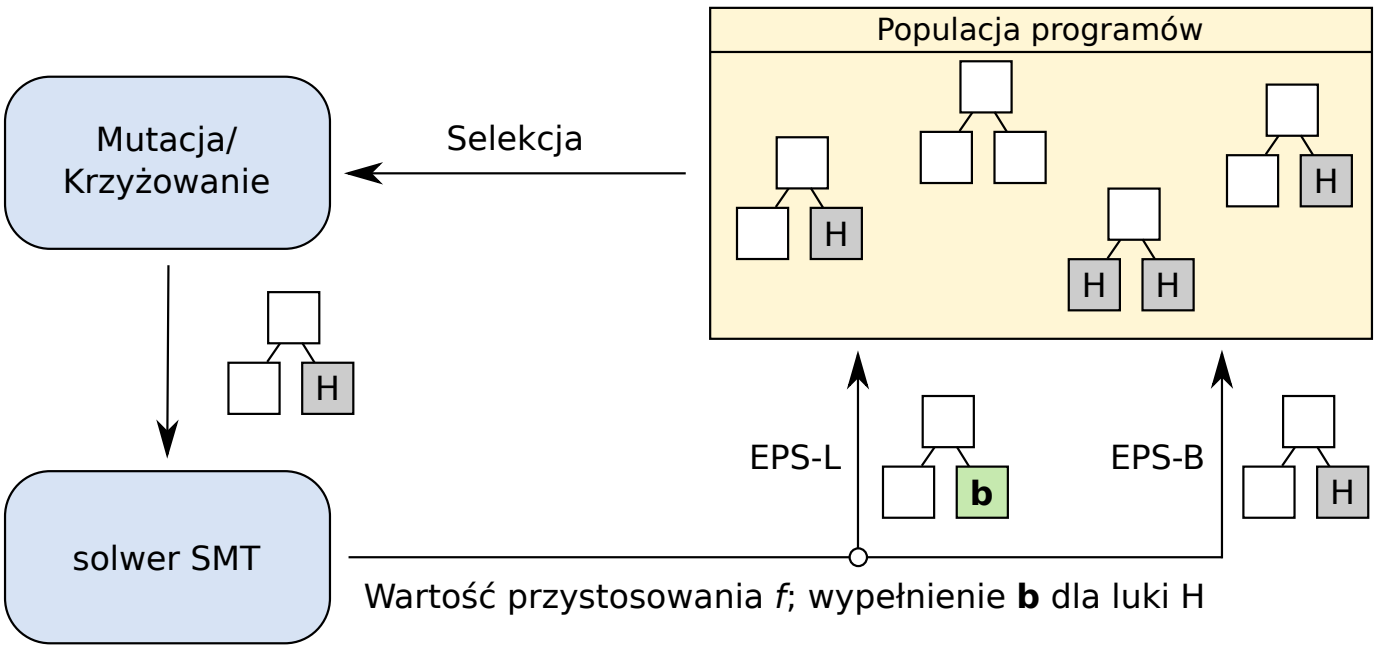
Zaproponowane rozwiązanie:

- Algorytm memetyczny w którym GP generuje szkice, które są następnie wypełniane przez solwer SMT.
- Solwer SMT użyty niestandardowo w roli optymalizatora, znajdując wypełnienie luk optymalizujące liczbę spełnionych testów.

[1] Armando Solar-Lezama et al., "Combinatorial Sketching for Finite Programs", 2006.

Ewolucyjne szkicowanie programów (EPS)

Diagram działania



- **EPS-L** — wariant „Lamarckowski”, w którym luki w programie są zastępowane przez znalezione optymalne ich wypełnienie.
- **EPS-B** — wariant „Baldwinowski”, w którym program pozostaje bez zmian, ale jego wartość przystosowania jest taka sama jak dla programu stworzonego w przypadku EPS-L.

Eksperymenty Obliczeniowe (EPS)

Cele:

- Sprawdzenie skuteczności opracowanej techniki dla problemu regresji symbolicznej.
- Porównanie wariantu Lamarckowskiego z Baldwinowskim.

Wnioski:

- Wypełnianie slotów tylko stałymi lub stałymi i zmiennymi okazało się być znacznie bardziej efektywne niż tylko zmiennymi.
- Wariant Baldwinowski (EPS-B) okazał się być zdecydowanie najbardziej skuteczny.
- Jednocześnie wariant ten był też najbardziej kosztowny obliczeniowo.
- Wyniki EPS-B pozostają znacznie lepsze niż w przypadku konfiguracji GP_T działającej przez średnią czasu EPS-B na danym benchmarku.

Plan prezentacji

GP kierowane kontrprzykładami (CDGP)

Problem badawczy:

- Jest niewiele prac łączących stochastyczne algorytmy optymalizacji globalnej z syntezą programów na podstawie formalnych specyfikacji (ograniczeń logicznych).
- Fundamentalna trudność: jak skonstruować informatywną miarę błędu kiedy kryterium poprawności jest zasadniczo binarne.
- Najprostsze rozwiązanie: zdekomponować kryterium poprawności. Jednak liczba rozróżnialnych poziomów jakości programów jest wtedy często niewielka.

Zaproponowane rozwiązanie:

- Weryfikacja programów wygenerowanych przez GP w celu zagwarantowania ich poprawności ze względu na logiczną specyfiakcję.
- Stopniowe dostarczanie GP kontrprzykładów zwróconych przez nieudane próby weryfikacji i obliczanie na ich podstawie wartości przystosowania o znacznie większej liczbie poziomów oceny jakości osobników.

Stan literatury

Podejścia do generowania przez GP programów z gwarancjami poprawności

Wartość przystosowania oparta na liczbie spełnionych ograniczeń

- (2007) C.G. Johnson, *Genetic Programming with Fitness Based on Model Checking*
- (2008) G. Katz, D. Peled, *Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms*
- (2011) P. He, L. Kang, C.G. Johnson, S. Ying, *Hoare logic-based genetic programming*
- (2016) G. Katz, D. Peled, *Synthesizing, correcting and improving code, using model checking-based genetic programming*

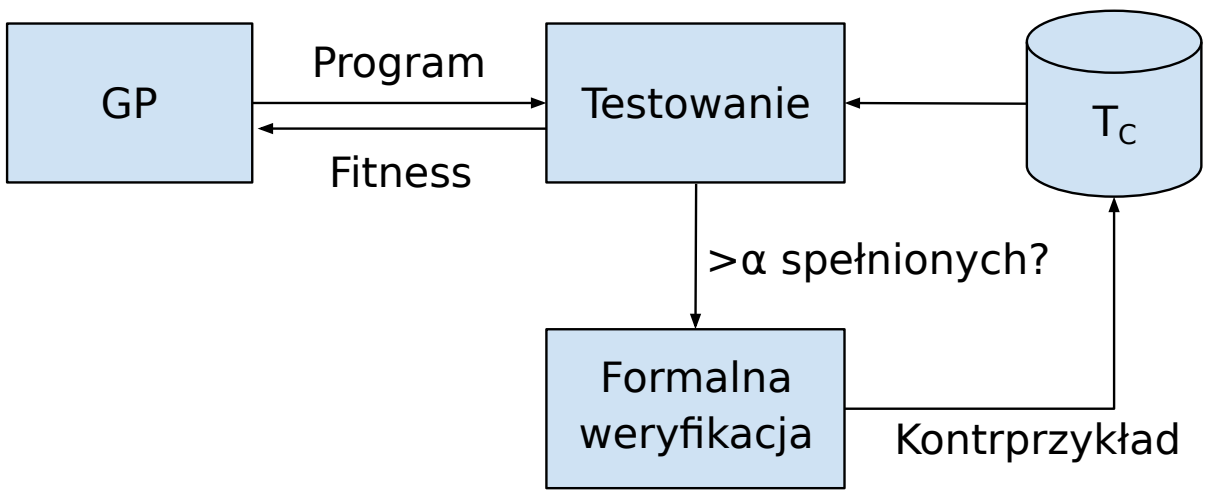
Generowanie kontrprzykładów

- (2017) I.Błądek, K.Krawiec, J.Swan, *Counterexample-Driven Genetic Programming: Heuristic Program Synthesis from Formal Specifications*

GP kierowane kontrprzykładami (CDGP)

CDGP składa się z trzech głównych modułów:

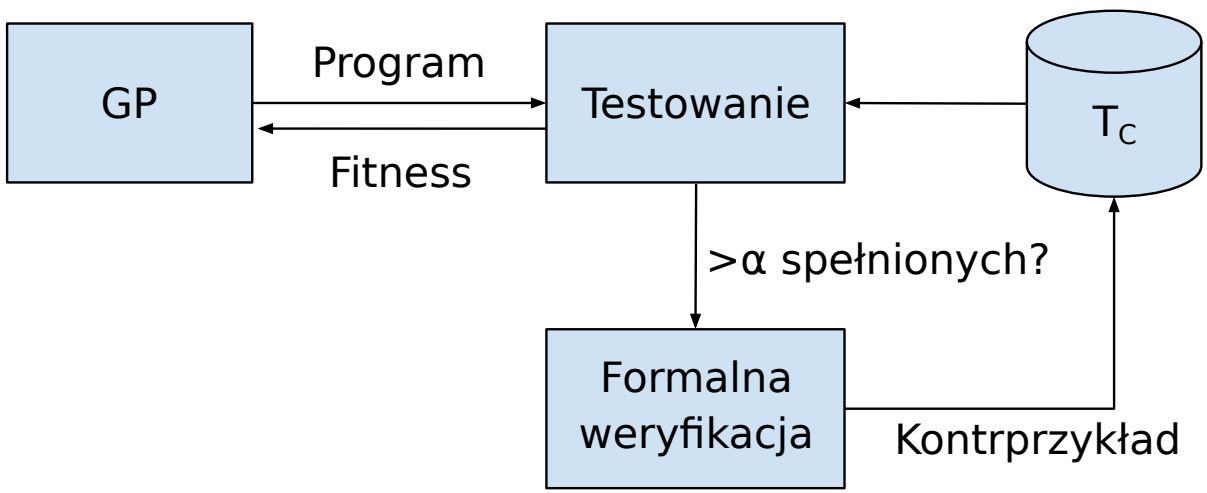
- **Algorytm przeszukiwania – GP**
- **Testowanie**
- **Weryfikacja**



GP kierowane kontrprzykładami (CDGP)

Moduł: **GP**

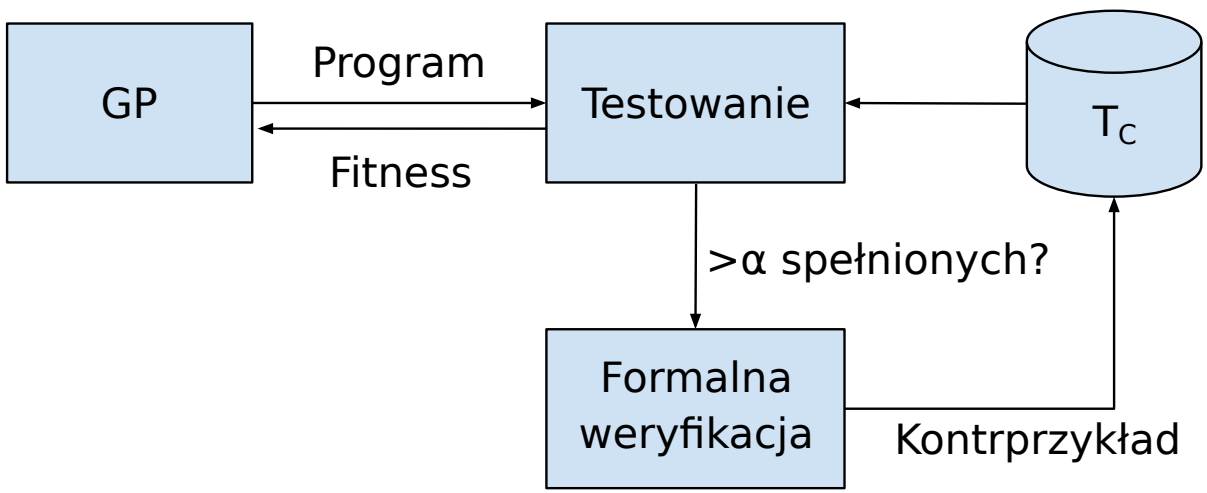
- Utrzymuje populację rozwiązań kandydackich, i stopniowo ją ulepsza korzystając z GP.
- Wymaga do działania wartości przystosowania rozwiązań kandydackich w populacji, które to wartości dostarczane są przez moduł **Testowania**.



GP kierowane kontrprzykładami (CDGP)

Moduł: **Testowanie**

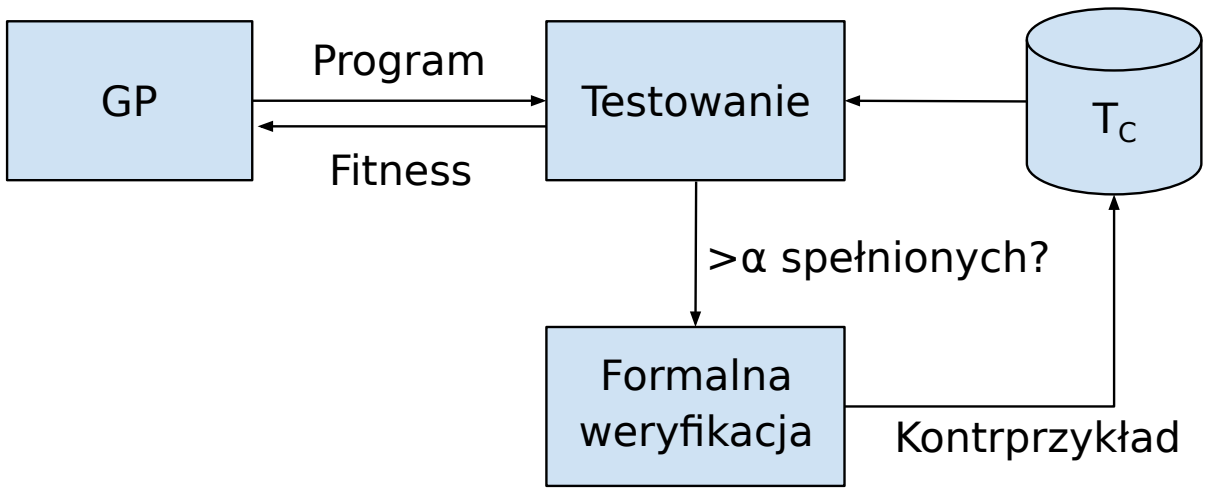
- Wartość przystosowania obliczana jest na podstawie testów zgromadzonych w zbiorze T_c.
- T_c na początku jest pusty lub zainicjalizowany testami podanymi przez użytkownika.
- Podczas fazy **Formalnej weryfikacji** do T_c mogą zostać dodane nowe testy.



GP kierowane kontrprzykładami (CDGP)

Moduł: **Formalna weryfikacja**

- Sprawdza, czy rozwiązanie spełnia ograniczenia logiczne.
- Jeżeli rozwiązanie nie spełnia ograniczeń, to znaleziony kontrprzykład jest konwertowany do przypadku testowego.
- Korzysta z solwera SMT.
- Jest wykonywane tylko jeżeli co najmniej α procent zebranych testów jest spełnionych (parametr algorytmu).



Rodzaje testów w CDGP

Testy w T_c można podzielić na dwa podzbiory:

- **Testy kompletne**
  Standardowe testy w formie (*wejście*, *wyjście*).

- **Testy niekompletne**
  Testy, dla których jest więcej niż jedno poprawne wyjście.
  Są one ewaluowane przez solwer SMT.

Przykład 1: $f(x) = \sqrt{x}$

x	$f(x)$
4	$2 \vee -2$
9	$3 \vee -3$
...	...

$\leftarrow$ Nie ma jednej wartości, z którą można by się porównać

Rodzaje testów w CDGP

Testy w T_c można podzielić na dwa podzbiory:

- **Testy kompletne**
  Standardowe testy w formie (*wejście*, *wyjście*).

- **Testy niekompletne**
  Testy, dla których jest więcej niż jedno poprawne wyjście.
  Są one ewaluowane przez solwer SMT.

Przykład 2: $f(x) > 2x$

x	$f(x)$
4	9, 10, ...
9	19, 20, ...
...	...

$\leftarrow$ Nie ma jednej wartości, z którą można by się porównać

Eksperymenty Obliczeniowe (CDGP)

Cele:

- Zbadanie efektywności CDGP dla różnych parametrów.
- Zbadanie efektywności CDGP dla problemów syntezy z różnych dziedzin.
- Porównanie z formalnymi algorytmami syntezy działających na podstawie formalnej specyfikacji.

Wnioski:

- Dla benchmarków LIA (liniowa arytmetyka liczb całkowitych) CDGP działa gorzej niż formalne metody syntezy (EUSolver, CVC4). Znacznie dłuższy czas działania i niższa skuteczność.
- Dla benchmarków SLIA (operacje na tekście z elementami LIA) to CDGP odniosło lepsze wyniki, rozwiązując więcej problemów.
- Dla obu problemów CDGP często znajdowało krótsze programy.
- Parametr $\alpha = 0.75$ okazał się być najbardziej skuteczny.
- Kontrprzykłady znalezione przez solwer SMT są znacznie lepsze niż losowo wygenerowane.

Plan prezentacji

Regresja symboliczna kierowana kontrprzykładami (CDSR)

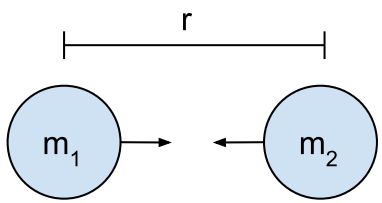
Uczenie nadzorowane z ograniczeniami

Dla zbioru uczącego T zawierającego n przykładów wejście-wyjście

$$(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_n}, y_n),$$

gdzie y_j zostały wygenerowane przez nieznaną funkcję f taką że $y_i = f(\boldsymbol{x_i})$, i dla danego zbioru ograniczeń C, znajdź taką funkcję h która będzie możliwie wiernie aproksymowała f jednocześnie spełniając wszystkie ograniczenia z C.

Przykład:



m_1	m_2	r	oczekiwany wynik
10.88386	11.36099	15.74871	0.00000000003327326009689590 5
11.1782	3.21214	7.67932	0.00000000004063563149853990 7
...			

przy ograniczeniach:

- $f(m_1, m_2, r) = f(m_2, m_1, r)$
- $f(m_1, m_2, r) \geq 0$
- silna monotoniczność f ze względu na m_1 i m_2

Przykładowe ograniczenia

Przykłady często wykorzystywanych ograniczeń:

- Zakres wartości funkcji w pewnym przedziale: $f(x, y) \in [0, 1]$
- Symetria ze względu na zamianę argumentów: $f(x, y) = f(y, x)$
- Symetria ze względu na znak argumentu: $f(x) = f(-x)$
- Monotoniczność: $\forall_{x,y}\ x > y \implies f(x) > f(y)$
- Wypukłość/wklęsłość
- Wartość pochodnej w punkcie: $f'(3.7) \geq 2.5$

W ogólności zależy nam na tym, żeby użytkownik mógł zdefiniować dowolne ograniczenie wyrażalne w wybranym formalizmie.

Naszą odpowiedzią na te potrzeby jest próba dostosowania CDGP do problemów regresji, w wyniku czego powstała **regresja symboliczna kierowana kontrprzykładami** (CDSR).

Zmiany w porównaniu do CDGP

	CDGP	**CDSR**
Kryterium weryfikacji	stosunek α spełnionych testów	stosunek α spełnionych testów; dla testów niekompletnych określony jest *próg spełnialności* (domyślnie 5% oczekiwanej wartości)
Zbiór walidacyjny	nie	tak
Kryterium stopu (poza limitem 1800 s na obliczenia)	program spełnia ograniczenia logiczne	brak poprawy na zbiorze walidacyjnym przez 25 pokoleń
Formalne ograniczenia uwzględnione bezpośrednio w wektorze przystosowania	nie	tak dla wariantu CDSR_p

Wariant $CDSR_p$

- Rozszerza zbiór testów T_c o nowe niekompletne testy odpowiadające spełnianiu pojedynczych ograniczeń.
- Ewaluowane przy użyciu solwera SMT.
- Największa korzyść przy użyciu selekcji ϵ-Lexicase[2].
- Dodatkowy parametr pozwala regulować relatywną istotność przykładów uczących i ograniczeń.

#	r1	r2	out
1	10.09611	17.39521	6.388341979690316
2	0.68719	4.75438	0.600408042568597
3	1.42871	17.19419	1.319102352206155
4	$f(r_1, r_2) = f(r_2, r_1)$		-
5	$r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}$		-
6	$f(r_1, r_2) \leq r_1 \quad \wedge \quad f(r_1, r_2) \leq r_2$		-

[2]W.L.Cava, L.Spector, K.Danai, "Epsilon-lexicase Selection for Regression", GECCO, 2016.

Eksperymenty Obliczeniowe (CDSR)

Cele:

- Zbadanie efektywności CDSR dla różnych parametrów.
- Zbadanie efektywności CDSR dla różnych rodzajów ograniczeń.
- Porównanie ze standardowymi algorytmami regresji jeżeli chodzi o błąd i spełnianie ograniczeń.

Wnioski:

- CDSR_p okazał się skuteczniejszy jeżeli chodzi o spełnianie ograniczeń niż CDSR, ale kosztem błędu na zbiorze testowym.
- Najlepsze standardowe algorytmy regresji spełniają średnio więcej ograniczeń, ale CDSR istotnie częściej daje radę spełnić je wszystkie na raz.
- Wykorzystanie solwera SMT generuje bardzo duży narzut czasowy.

Plan prezentacji

GP wspierane sztuczną siecią neuronową (NGGP)

Problem badawczy:

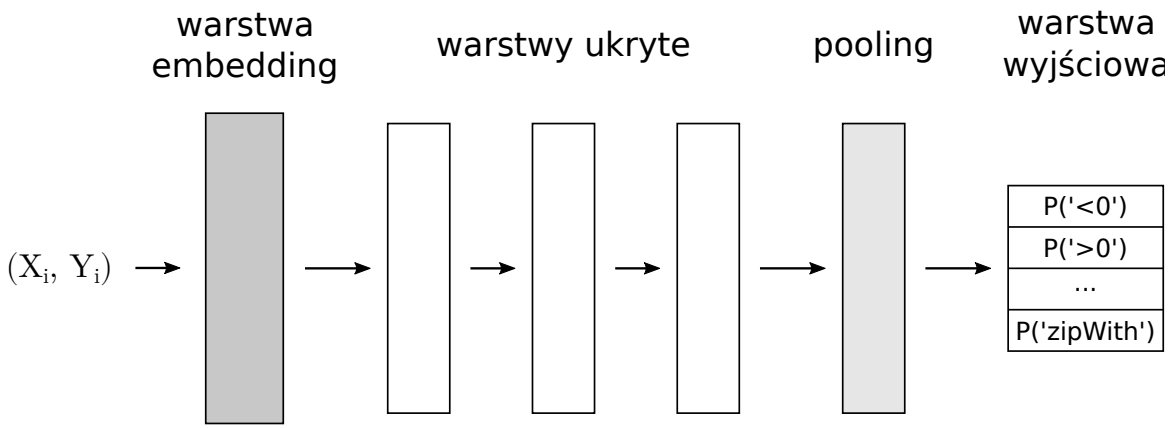
- Przeszukiwanie przestrzeni rozwiązań może być usprawnione poprzez mechanizm priorytetyzacji, to znaczy przeszukiwania najpierw obiecujących fragmentów przestrzeni rozwiązań.
- Tego typu podejście, bazujące na uczeniu maszynowym, zostało wcześniej opisane przez Matej Balog et al. i znane jest jako DeepCoder[3].
- Pytanie brzmi, czy technika ta jest transferowalna do domeny obliczeń ewolucyjnych, oraz jak duża jest kompetytywna przewaga z niej wynikająca.

Zaproponowane rozwiązanie:

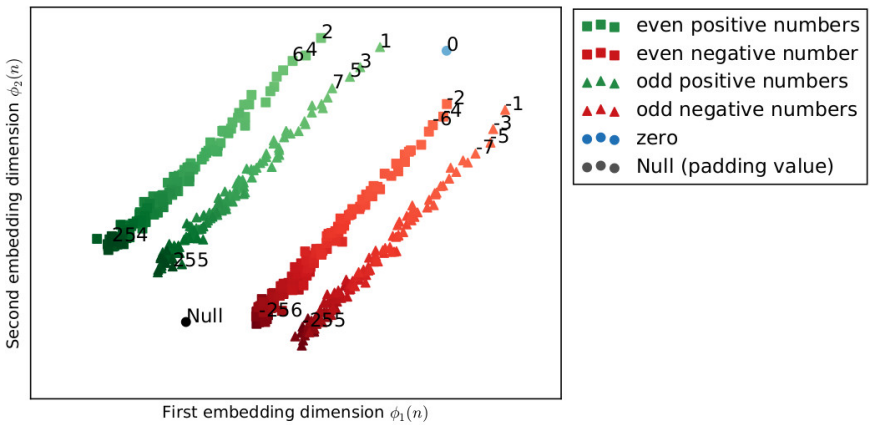
- Opracowanie wariantu liniowego GP ze zmodyfikowanymi operatorami mutacji i inicjalizacji uwzględniającymi informację zwróconą przez model uczenia maszynowego.
- Model uczenia maszynowego to, tak samo jak w przypadku DeepCodera, sieć neuronowa uczona na zbiorze instancji danego problemu syntezy.

[3] Matej Balog, et al., "DeepCoder: Learning to Write Programs", 2017.

Architektura sieci neuronowej



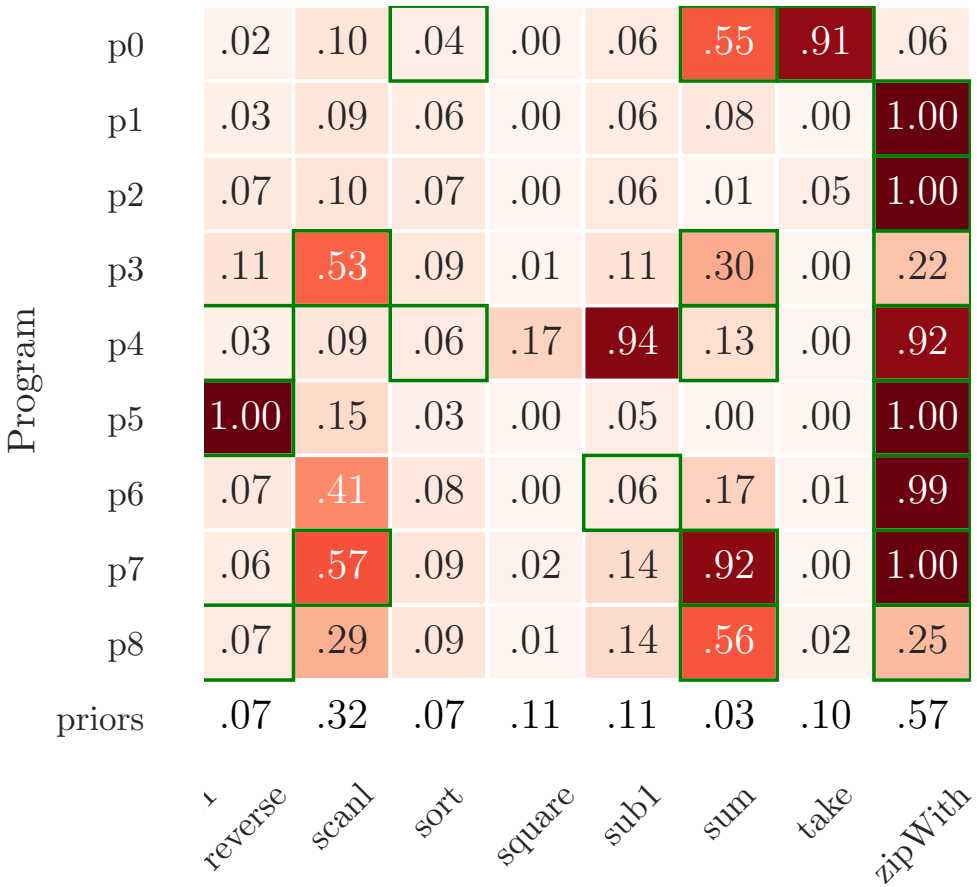
Celem wykorzystania warstwy "embedding" (pol. wektory osadzone) jest nadanie głębszej semantyki stałym języka ze względu na zbiór instrukcji programu.



Matej Balog, et al., "DeepCoder: Learning to Write Programs", 2017.

Warstwa "pooling" (pol. warstwa łącząca) agreguje wynik indywidualnego przetwarzania wszystkich przykładów ze specyfikacji przez warstwy ukryte.

Wizualizacja działania sieci neuronowej



P0: Oblicz sumę *a* najmniejszych elementów na liście *b*.

Poprawny program:

```
1 a ← int
2 b ← [int]
3 c ← Sort b
4 d ← Take a c
5 e ← Sum d
```

Specyfikacja:

```
2, [1 5 3]   → 4
1, [1 8 3 5] → 1
3, [1 8 3 5 7] → 9
...
```

Eksperymenty Obliczeniowe (NGGP)

Cele:

- Zbadanie przydatności sieci neuronowej do priorytetyzacji przeszukiwania w GP.

Wnioski:

- Istotna poprawna efektywności działania GP wspieranego siecią neuronową w porównaniu do wariantów bez takiego wsparcia lub z prostym obciążeniem częstościowym.
- Użycie priorytetyzacji także podczas inicjalizacji pozwoliło uzyskać znacznie lepszy rezultat niż wyłącznie podczas mutacji.

Plan prezentacji

Podsumowanie

- Opracowanie metod wzbogacania programowania genetycznego dodatkową informacją w celu poprawy jego skuteczności i szybkości działania dla problemów syntezy programów.
  - **EPS — najlepsze wypełnienie luk programu.**
  - **CDGP/CDSR — wzbogacanie kontrprzykładami.**
  - **NGGP — nauczony model uczenia maszynowego.**
- Stworzenie ewolucyjnych algorytmów syntezy programów z gwarancjami poprawności.
  - **CDGP/CDSR — gwarancje uzyskane przez dowody w SMT.**
- Analiza zdolności uogólniania w przypadku specyfikacji formalnej określonej zbiorem ograniczeń logicznych.
  - **CDGP/CDSR — kryteria liczby spełnionych indywidualnych ograniczeń i procent uruchomień, w których spełniono wszystkie ograniczenia.**
- Eksperymentalna ocena algorytmów na zbiorach benchmarków znanych z literatury, a tam gdzie ich brak utworzenie własnych benchmarków na podstawie problemów zaczerpniętych z dziedzin m.in. programowania, matematyki i fizyki.
  - **CDSR/CDSR — konieczność opracowania własnych benchmarków.**

Koniec części głównej prezentacji

Dziękuję za uwagę.

Plan prezentacji

Pytania – dr hab. Wojciech Jamroga, prof. IPI PAN

Rozdział 5: byłoby interesujące porównać wyniki z efektywnością podejścia ewolucyjnego z syntezą programów za pomocą „czystego” SMT.

Moje własne eksperymenty z syntezą SMT wskazywały na bardzo długi czas obliczeń związany z tą technologią. Używałem jednak tej techniki bezpośrednio według definicji logicznej poprawności, podczas gdy odpowiednie kodowanie problemu mogłoby potencjalnie dać lepszy efekt. Podejrzewam, że EPS, pomimo odsunięcia od użytkownika konieczności definiowania szkicu, okazałby się mniej efektywny w praktyce.

Pytania – dr hab. Wojciech Jamroga, prof. IPI PAN

Str. 64: Jako że specyfikacje wejścia/wyjścia są (na poziomie logicznym) po prostu bardzo długimi koniunkcjami prostych predykatów, zaprezentowana metoda sprowadza się do zliczania ilości poprawnych przypadków testowych. W takim ujęciu funkcja zdrowia nie wydaje się być binarna.

Zgadza się, ekspozycja materiału mogła być lepsza w tamtym punkcie rozprawy. Tak naprawdę zbiór przypadków testowych stanowi pewną formułą logiczną, którą możemy łatwo zdekomponować na części pośrednie. Co istotne, zazwyczaj otrzymamy w ten sposób liczny zbiór indywidualnych testów, i w konsekwencji dobrą miarę funkcji przystosowania. W przypadku specyfikacji formalnej, nawet po rozkładzie do postaci CNF, tego typu miara nie dostarczy dużego stopniowania jakości rozwiązań.

Pytania – dr hab. Wojciech Jamroga, prof. IPI PAN

Rozdział 8: czy zaprezentowane tu podejście nie mogłoby dać ciekawych wyników w połączeniu z metodami prezentowanymi w poprzednich rozdziałach?

Tak, nie ma absolutnie żadnych przeszkód żeby coś takiego zrobić. Metoda przedstawiona w rozdziale 8. jest zasadniczo uniwersalna, aczkolwiek detale implementacji sieci neuronowej mogą być znacznie różne w zależności od rodzaju przetwarzanych danych i języka programowania.

Pytania – prof. dr hab. inż. Robert Schaefer

Autor zakłada, że każdy z rozważanych problemów syntezy posiada przynajmniej jedno rozwiązanie. Spełnienie tego warunku ma umożliwić odpowiednio duża moc opisowa użytego języka programowania.

a) Czy rozważane problemy mogą posiadać więcej niż jedno rozwiązanie?
b) Czy rozważane strategie syntezy dają możliwość znalezienia więcej niż jednego rozwiązania lub wszystkich rozwiązań w takim przypadku?
c) Czy znajdowanie większej ilości rozwiązań takich problemów może być umotywowane praktycznie?

Uwaga wstępna: moją intencją było wskazanie w ten sposób zakresu stosowalności metod. Nie są one zoptymalizowane na sytuację, kiedy rozwiązanie poprawne nie istnieje. Nie mają zdolności wykrycia tego faktu i będą działać tak długo jak im pozwoli budżet obliczeniowy. Oczywiście nie ma żadnych przeciwwskazań, by uruchomić algorytm syntezy dla tego typu problemu. W tym sensie zgadzam się, że może nie powinienem tego nazywać "założeniem", a raczej po prostu "własnością" algorytmu syntezy.

Pytania – prof. dr hab. inż. Robert Schaefer

Autor zakłada, że każdy z rozważanych problemów syntezy posiada przynajmniej jedno rozwiązanie. Spełnienie tego warunku ma umożliwić odpowiednio duża moc opisowa użytego języka programowania.

a) Czy rozważane problemy mogą posiadać więcej niż jedno rozwiązanie?
b) Czy rozważane strategie syntezy dają możliwość znalezienia więcej niż jednego rozwiązania lub wszystkich rozwiązań w takim przypadku?
c) Czy znajdowanie większej ilości rozwiązań takich problemów może być umotywowane praktycznie?

---

a) Jak najbardziej tak. W przypadku większości rozpatrywanych w praktyce języków programowania, liczba programów o innym kodzie źródłowym ale semantycznie redundantnych jest bardzo duża. Jeżeli specyfikacja zadania nie definiuje poprawnego wyjścia dla każdego możliwego wejścia, to dodatkowo możemy wyróżnić szerszą klasę programów identycznych z perspektywy tej specyfikacji.

Pytania – prof. dr hab. inż. Robert Schaefer

Autor zakłada, że każdy z rozważanych problemów syntezy posiada przynajmniej jedno rozwiązanie. Spełnienie tego warunku ma umożliwić odpowiednio duża moc opisowa użytego języka programowania.

a) Czy rozważane problemy mogą posiadać więcej niż jedno rozwiązanie?
b) Czy rozważane strategie syntezy dają możliwość znalezienia więcej niż jednego rozwiązania lub wszystkich rozwiązań w takim przypadku?
c) Czy znajdowanie większej ilości rozwiązań takich problemów może być umotywowane praktycznie?

---

b) Bez modyfikacji strategii syntezy, znalezienie większej liczby rozwiązań można osiągnąć wyłącznie poprzez wielokrotne uruchomienie strategii dla różnych wartości ziarna algorytmu pseudolosowego (podstawa praktycznych implementacji stochastycznych algorytmów). Nigdy nie uzyskamy jednak gwarancji, że znaleźliśmy wszystkie rozwiązania semantycznie redundantne. Można skonstruować strategie syntezy o takich własnościach, ale te opracowane w mojej rozprawie nie były z tą myślą tworzone.

Pytania – prof. dr hab. inż. Robert Schaefer

Autor zakłada, że każdy z rozważanych problemów syntezy posiada przynajmniej jedno rozwiązanie. Spełnienie tego warunku ma umożliwić odpowiednio duża moc opisowa użytego języka programowania.

a) Czy rozważane problemy mogą posiadać więcej niż jedno rozwiązanie?
b) Czy rozważane strategie syntezy dają możliwość znalezienia więcej niż jednego rozwiązania lub wszystkich rozwiązań w takim przypadku?
c) Czy znajdowanie większej ilości rozwiązań takich problemów może być umotywowane praktycznie?

---

c) Tak. Przykładowo, kiedyś podczas rozmowy z kolegą pojawiła się kwestia znalezienia wszystkich semantycznie unikalnych miar konfirmacji spełniających pewne warunki. Innym przykładem mogłoby być znajdowanie dokładnie wszystkich elementów frontu Pareto w przypadku optymalizacji wielokryterialnej.

Pytania – prof. dr hab. inż. Robert Schaefer

Jaki może być stosunek kosztu (czasu CPU, RAM, etc.) etapów wstępnych, każdej z proponowanych metod do uzyskanego przyspieszenia (lub innej formy poprawy działania) dla pojedynczej syntezy w stosunku do GP bez uzupełniania wiedzą?

Zawężę się do czasu CPU, ponieważ zużycie pamięci jest relatywnie niewielkie w przypadku prezentowanych metod.

a) EPS — ten przykład pokazuje nam, że przyspieszenie to nie wszystko, bo istotna jest również skuteczność. Przetestowany w rozprawie wariant GP_T o takim samym budżecie procesora jak średnie uruchomienie EPS osiągnął znacznie gorszy wynik jeżeli chodzi o skuteczność syntezy (w przypadku benchmarku Koza1, na którym uzyskał najwyższy wynik, było to 0.68 do 1.0 w przypadku wariantu Baldwinowskiego).

Pytania – prof. dr hab. inż. Robert Schaefer

Jaki może być stosunek kosztu (czasu CPU, RAM, etc.) etapów wstępnych, każdej z proponowanych metod do uzyskanego przyspieszenia (lub innej formy poprawy działania) dla pojedynczej syntezy w stosunku do GP bez uzupełniania wiedzą?

---

b) CDGP — brak możliwości porównania, ponieważ czyste GP nie jest w stanie rozwiązać problemu opartego wyłącznie na formalnej specyfikacji.

c) CDSR — od przypadku wyżej odróżnia nas to, że GP może działać na podstawie zbioru testów ignorując bezpośrednią informację o ograniczeniach (jest ona jednak zawarta *pośrednio* w testach). Spodziewam się, że stosunek kosztu związany z poprawą liczby spełnionych ograniczeń byłby relatywnie wysoki.

Pytania – prof. dr hab. inż. Robert Schaefer

Jaki może być stosunek kosztu (czasu CPU, RAM, etc.) etapów wstępnych, każdej z proponowanych metod do uzyskanego przyspieszenia (lub innej formy poprawy działania) dla pojedynczej syntezy w stosunku do GP bez uzupełniania wiedzą?

---

d) NGGP — wykorzystanie wiedzy pozyskanej w tym przypadku jest powiązane z istotnym stałym początkowym kosztem CPU, i pomijalnym kosztem podczas pojedynczego uruchomienia. Dlatego też „pojedyncza synteza" to moim zdaniem nieodpowiednie kryterium dla NGGP.