Arytmetyka Presburgera

Rozstrzygalność, złożoność i zastosowania

Michał Dobranowski

Akademia Górniczo-Hutnicza w Krakowie

Kraków, 12 kwietnia 2026

Kilka definicji

Sygnatura i struktura

Sygnatura to zbiór symboli funkcyjnych (np. $+$, $\cdot$), symboli relacyjnych (np. $<$, $|$) oraz symboli stałych (np. 0, 1).

Sygnatura i struktura

Sygnatura to zbiór symboli funkcyjnych (np. $+$, $\cdot$), symboli relacyjnych (np. $<$, $|$) oraz symboli stałych (np. 0, 1).

Struktura to pewien zbiór A wraz z sygnaturą oraz interpretacją jej symboli. Strukturą jest np.

$$\langle \mathbb{N}; 0, 1, +, \cdot, < \rangle.$$

wraz z naturalną interpretacją poszczególnych symboli.

Teoria struktury

Teoria struktury $\mathfrak{M}$ to zbiór zdań prawdziwych w logice pierwszego rzędu w tej strukturze.

Teoria struktury

Teoria struktury $\mathfrak{M}$ to zbiór zdań prawdziwych w logice pierwszego rzędu w tej strukturze.

Na przykład, jeśli weźmiemy $\mathfrak{M} = \langle \mathbb{N}; +, < \rangle$, to

$$(\forall x \exists y : x < y) \in \mathrm{Th}(\mathfrak{M}),$$

Teoria struktury

Teoria struktury $\mathfrak{M}$ to zbiór zdań prawdziwych w logice pierwszego rzędu w tej strukturze.

Na przykład, jeśli weźmiemy $\mathfrak{M} = \langle \mathbb{N}; +, < \rangle$, to

$$(\forall x \exists y : x < y) \in \mathrm{Th}(\mathfrak{M}),$$

$$(\exists x \forall y : y < x + x) \notin \mathrm{Th}(\mathfrak{M}),$$

Teoria struktury

Teoria struktury $\mathfrak{M}$ to zbiór zdań prawdziwych w logice pierwszego rzędu w tej strukturze.

Na przykład, jeśli weźmiemy $\mathfrak{M} = \langle \mathbb{N}; +, < \rangle$, to

$$(\forall x \exists y : x < y) \in \mathrm{Th}(\mathfrak{M}),$$

$$(\exists x \forall y : y < x + x) \notin \mathrm{Th}(\mathfrak{M}),$$

$$(\forall x \exists y : x < y^y) \notin \mathrm{Th}(\mathfrak{M}).$$

Arytmetyka Presburgera

Z twierdzenia Churcha wynika, że $\mathrm{Th}\langle \mathbb{N}; 0, 1, +, \cdot, = \rangle$ jest nierozstrzygalna.

Jeśli jednak usuniemy mnożenie, to otrzymamy teorię $\mathrm{Th}\langle \mathbb{N}; 0, 1, +, = \rangle$, zwaną *arytmetyką Presburgera*, która **jest rozstrzygalna**.

Siła wyrazu

Możemy wyrażać:

Siła wyrazu

Możemy wyrażać:

- mnożenie przez stałą z $\mathbb{N}$

$$3a \equiv a + a + a$$

Siła wyrazu

Możemy wyrażać:

- mnożenie przez stałą z $\mathbb{N}$

$$3a \equiv a + a + a$$

- podzielność przez stałą z $\mathbb{N}$

$$2 \mid n \equiv \exists a : 2a = n$$

Siła wyrazu

Możemy wyrażać:

- mnożenie przez stałą z $\mathbb{N}$

$$3a \equiv a + a + a$$

- podzielność przez stałą z $\mathbb{N}$

$$2 \mid n \equiv \exists a : 2a = n$$

- nierówności

$$a < b \equiv \exists c : a + c + 1 = b$$

Rozstrzygalność

Eliminacja kwantyfikatorów

W ogólności z formuły zbudowanej nad sygnaturą $\{0, 1, +, =\}$ nie możemy wyeliminować kwantyfikatorów. 😢

Eliminacja kwantyfikatorów

W ogólności z formuły zbudowanej nad sygnaturą $\{0, 1, +, =\}$ nie możemy wyeliminować kwantyfikatorów. 😢

Ale jeśli dołożymy do sygnatury relację nierówności i relację podzielności (nie zmieniając siły wyrazu języka!), to już możemy. 😊

Algorytm eliminacji kwantyfikatorów

1. Zmieniamy kwantyfikatory $\forall$ na $\exists$.

$$\forall x : \neg \square \quad \equiv \quad \neg \exists x : \square$$

[1] da się to zrobić!

Algorytm eliminacji kwantyfikatorów

1. Zmieniamy kwantyfikatory $\forall$ na $\exists$.

$$\forall x : \neg \square \equiv \neg \exists x : \square$$

2. Sprowadzamy formułę do *dysjunkcyjnej postaci normalnej* (alternatywy koniunkcji formuł atomowych), przy okazji usuwając wszelkie równości i zaprzeczenia[1].

$$\exists x : \square \equiv \exists x : (\square \vee \square \vee \square)$$

[1] da się to zrobić!

Algorytm eliminacji kwantyfikatorów

$$y = z \equiv y \not< z \wedge z \not< y$$

$$y \neq z \equiv y < z \vee z < y$$

$$y \not< z \equiv z < y + 1$$

$$k \nmid y \equiv \bigvee_{i=1}^{k-1} k \mid (y + i)$$

Algorytm eliminacji kwantyfikatorów

3. Dystrybuujemy kwantyfikator $\exists$ do poszczególnych części.

$$\exists x : (\blacksquare \vee \blacksquare \vee \blacksquare) \equiv (\exists x : \blacksquare) \vee (\exists x : \blacksquare) \vee (\exists x : \blacksquare)$$

Algorytm eliminacji kwantyfikatorów

3. Dystrybuujemy kwantyfikator $\exists$ do poszczególnych części.

$$\exists x : (\square \vee \square \vee \square) \equiv (\exists x : \square) \vee (\exists x : \square) \vee (\exists x : \square)$$

Każda z formuł $\square$ jest już tylko koniunkcją formuł atomowych (używających tylko jednego symbolu relacyjnego: $|$ lub $<$).

$$\exists x : \square = \exists x : \square \wedge \square \wedge \square \wedge \square$$

Algorytm eliminacji kwantyfikatorów

4. Zmieniamy odpowiednie formuły atomowe tak, żeby zmienna x miała zawsze ten sam współczynnik.

$$\exists x : ((3y < 5z + 2x) \wedge (2z < x + 1) \wedge (x < 10y) \wedge (4 \mid 3x)) \equiv$$
$$\equiv \exists x : ((9y < 15z + 6x) \wedge (12z < 6x + 6) \wedge (6x < 60y) \wedge (8 \mid 6x)) \equiv$$

Algorytm eliminacji kwantyfikatorów

4. Zmieniamy odpowiednie formuły atomowe tak, żeby zmienna x miała zawsze ten sam współczynnik.

$$\exists x : ((3y < 5z + 2x) \wedge (2z < x + 1) \wedge (x < 10y) \wedge (4 \mid 3x)) \equiv$$
$$\equiv \exists x : ((9y < 15z + 6x) \wedge (12z < 6x + 6) \wedge (6x < 60y) \wedge (8 \mid 6x)) \equiv$$

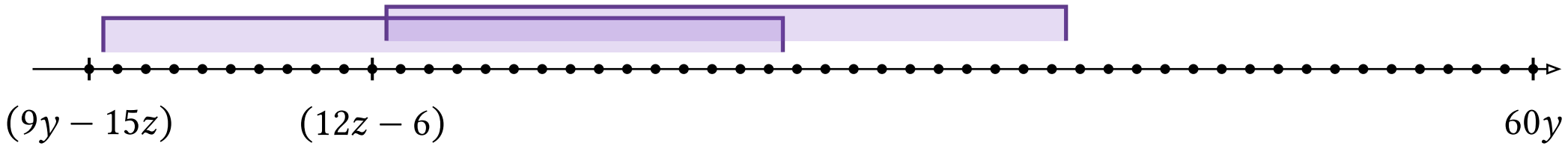
5. Pozbywamy się współczynnika przy zmiennej x.

$$\equiv \exists x : ((9y < 15z + x) \wedge (12z < x + 6) \wedge (x < 60y) \wedge (8 \mid x) \wedge (6 \mid x))$$

Algorytm eliminacji kwantyfikatorów

6. Na mocy CRT, rozwiązania części formuły związanej z podzielnością powtarzają się co $N = \operatorname{lcm}(8, 6) = 24$. Każdą z nierówności, w których x znajduje się po prawej stronie, zamieniamy na alternatywę N formuł.

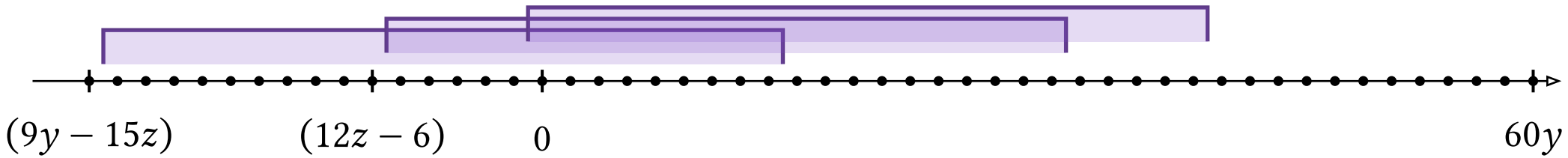
$$\exists x : ((9y < 15z + x) \wedge (12z < x + 6) \wedge (x < 60y) \wedge (8 \mid x) \wedge (6 \mid x))$$



Algorytm eliminacji kwantyfikatorów

6. Na mocy CRT, rozwiązania części formuły związanej z podzielnością powtarzają się co $N = \operatorname{lcm}(8, 6) = 24$. Każdą z nierówności, w których x znajduje się po prawej stronie, zamieniamy na alternatywę N formuł. Dla pewności dodajemy też ograniczenie dolne $0 \leq x$.

$$\exists x : ((9y < 15z + x) \wedge (12z < x + 6) \wedge (x < 60y) \wedge (8 \mid x) \wedge (6 \mid x))$$



Algorytm eliminacji kwantyfikatorów

$$\exists x : \underbrace{((9y < 15z + x) \wedge (12z < x + 6) \wedge (x < 60y) \wedge (8 \mid x) \wedge (6 \mid x))}_{\Phi(x,y,z)} \equiv$$

$$\equiv \bigvee_{j=1}^{N} \Phi(9y - 15z + j, y, z) \vee \bigvee_{j=1}^{N} \Phi(12z - 6 + j, y, z) \vee \bigvee_{j=0}^{N-1} \Phi(j, y, z)$$

Pozbyliśmy się kwantyfikatorów!

Arytmetyka Presburgera jest rozstrzygalna!

Złożoność konstrukcji

W kroku 2. należy zmienić formułę logiczną do DNF. Jest to problem NP -trudny, a rozmiar formuły w DNF może być wykładniczy względem oryginalnej formuły.

Złożoność konstrukcji

W kroku 2. należy zmienić formułę logiczną do DNF. Jest to problem NP -trudny, a rozmiar formuły w DNF może być wykładniczy względem oryginalnej formuły.

W kroku 4. używamy lcm(...) względem współczynników x.

Złożoność konstrukcji

W kroku 2. należy zmienić formułę logiczną do DNF. Jest to problem NP -trudny, a rozmiar formuły w DNF może być wykładniczy względem oryginalnej formuły.

W kroku 4. używamy lcm(...) względem współczynników x.

W kroku 6. ponownie używamy lcm(...), tym razem względem liczb, przez które sprawdzamy podzielność.

Złożoność konstrukcji

W kroku 2. należy zmienić formułę logiczną do DNF. Jest to problem NP -trudny, a rozmiar formuły w DNF może być wykładniczy względem oryginalnej formuły.

W kroku 4. używamy lcm(...) względem współczynników x.

W kroku 6. ponownie używamy lcm(...), tym razem względem liczb, przez które sprawdzamy podzielność.

Wykonujemy te kroki dla każdego bloku naprzemiennych kwantyfikatorów!

Dowód Büchiego

Idea dowodu

Zamiast eliminacji kwantyfikatorów, można zbudować DFA, który akceptuje język reprezentujący wszystkie wartościowania zmiennych, dla których dana formuła jest prawdziwa.

Liczby naturalne zapisujemy w postaci binarnej (*little-endian*):

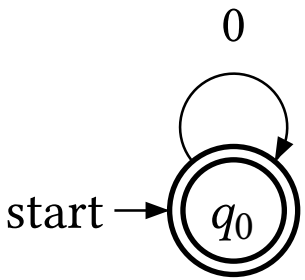
$$\mathbb{N} \cong \left(\{0,1\}^* 1\right) \cup \{0\}.$$

Zostawmy poprzednie rozszerzenia, wróćmy do podstawowej sygnatury arytmetyki Presburgera: $\{0, 1, +, =\}$.

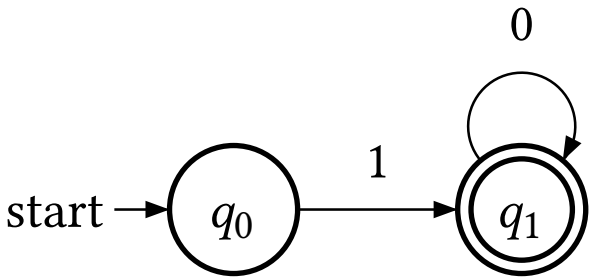
Pokażemy, że każdy symbol możemy zasymulować jakimś DFA.

Symbole funkcyjne, relacyjne i stałe

Automat rozpoznający $k = 0$:

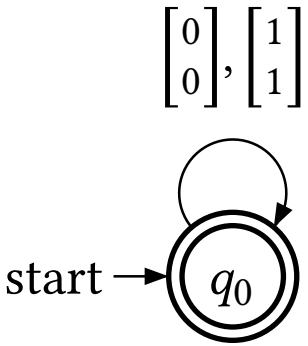


Automat rozpoznający $k = 1$:



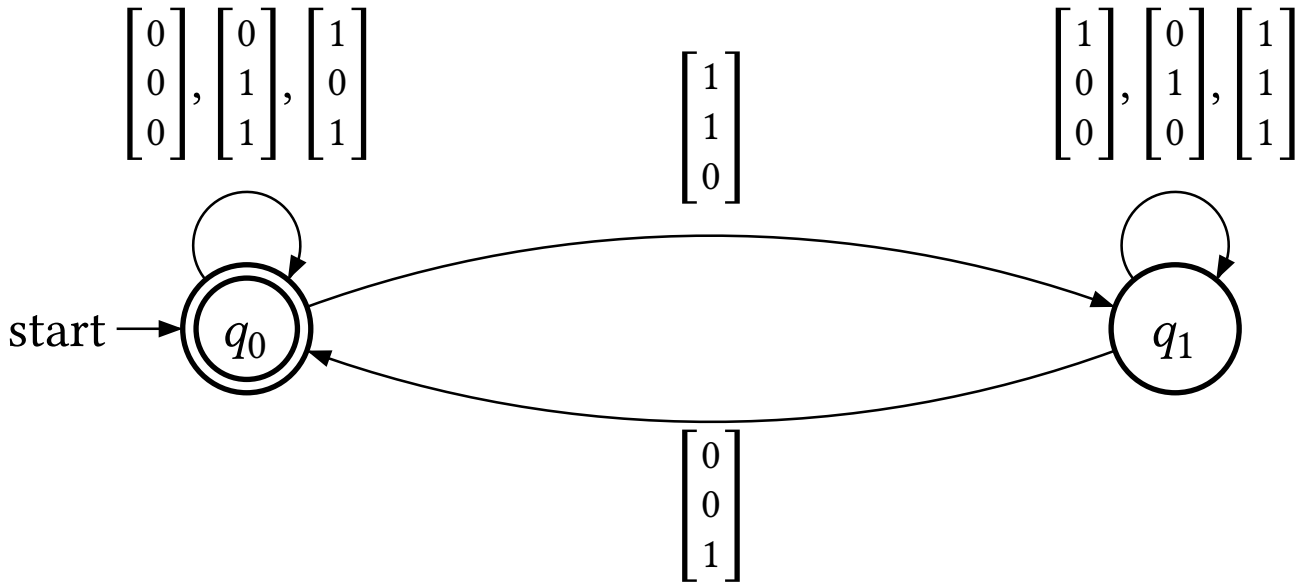
Symbole funkcyjne, relacyjne i stałe

Automat rozpoznający $i = j$:



Symbole funkcyjne, relacyjne i stałe

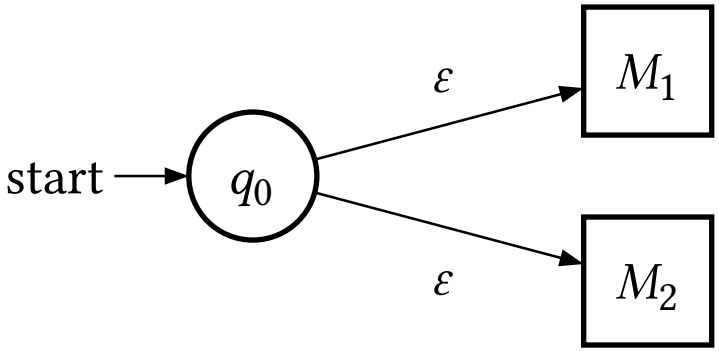
Automat rozpoznający $i + j = k$:



Analogicznie konstruujemy automaty akceptujące $i + 1 = j$, $i + j + k = \ell$, itd.

Spójniki logiczne

- koniunkcja ($\wedge$) – automat $M_1 \cap M_2$ tworzymy przez iloczyn kartezjański stanów automatów M_1, M_2,
- alternatywa ($\vee$) – niedeterministyczny automat $M_1 \cup M_2$ tworzymy jak poniżej,

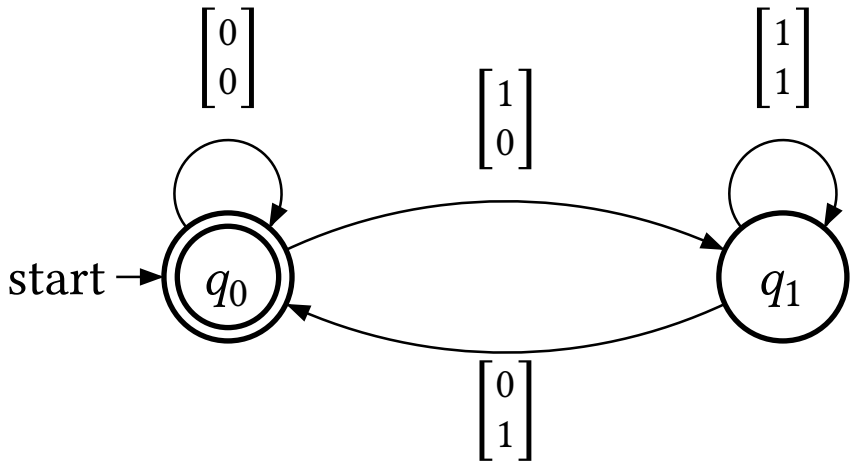


- negacja ($\neg$) – automat $M^{\complement}$ tworzymy z automatu zupełnego M przez zamianę stanów akceptujących i nieakceptujących.

Kwantyfikatory

Tak jak poprzednio, $\forall$ zamieniamy na $\exists$. Każdą zmienną związaną z $\exists$ wystarczy usunąć (będziemy ją *zgadywać* w ramach NFA).

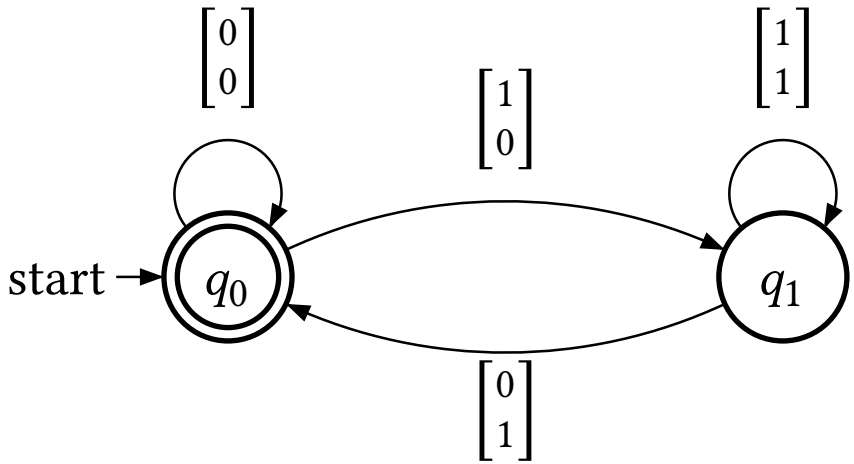
$j + j = k$



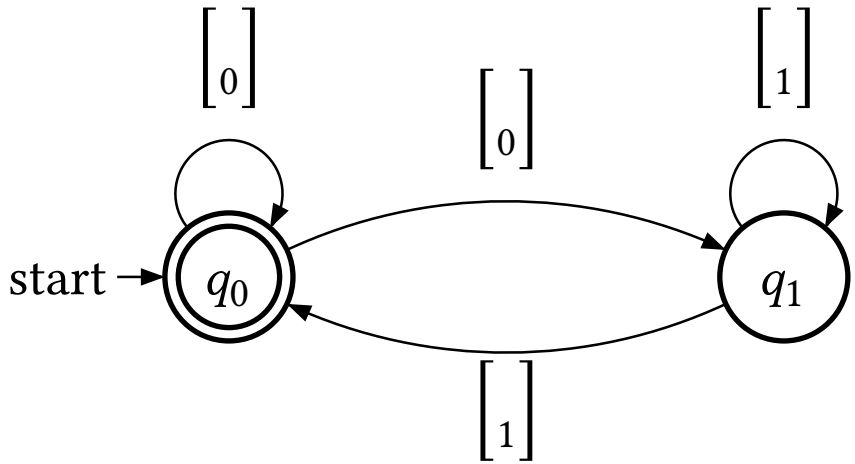
Kwantyfikatory

Tak jak poprzednio, $\forall$ zamieniamy na $\exists$. Każdą zmienną związaną z $\exists$ wystarczy usunąć (będziemy ją *zgadywać* w ramach NFA).

$j + j = k$



$\exists j : j + j = k$



Złożoność konstrukcji

Każda determinizacja NFA (powstałego z $\vee$ lub $\exists$) jest **wykładnicza**
(a determinizować musimy, żeby konstruować $\neg$).

Trudność arytmetyki Presburgera

Wyniki teoretyczne

- PrA jest między 2-NEXP a 2-EXPSPACE,
- dla każdego $i > 0$ fragment Σ_{i+1} PrA jest zupełny w Σ_i^{EXP}, w szczególności fragment Σ_2 PrA jest NEXP-zupełny.

Współczesne algorytmy

Na szczęście, mamy jakieś postępy w algorytmice:

- algorytm Coopera,
- test omega Pugha.

Zastosowania

Optymalizacja w czasie kompilacji

Kompilator (np. języka C) chce, żeby wiele iteracji jednej pętli było wykonywanych równocześnie. Czy może?

Optymalizacja w czasie kompilacji

Kompilator (np. języka C) chce, żeby wiele iteracji jednej pętli było wykonywanych równocześnie. Czy może?

To zależy od pętli.

```
for (int i = 0; i < n; i++) {
    A[2 * i] = A[i] + 1;
}
```

Optymalizacja w czasie kompilacji

Kompilator (np. języka C) chce, żeby wiele iteracji jednej pętli było wykonywanych równocześnie. Czy może?

To zależy od pętli.

```
for (int i = 0; i < n; i++) {
    A[2 * i] = A[i] + 1;
}
```

$$\exists i \exists j : \Big(i < n \land j < n \land i \neq j \land \big(\underbrace{2i = 2j}_{\text{zapis-zapis}} \lor \underbrace{2i = j}_{\text{zapis-odczyt}} \lor \underbrace{i = 2j}_{\text{odczyt-zapis}} \big) \Big)$$

Formuła prawdziwa dla $n > 2$, więc istnieje konflikt, więc kompilator nie może nic zrobić.

Optymalizacja w czasie kompilacji

```
for (int i = 0; i < n; i++) {
    A[3 * i + 5] = A[3 * i + 7] + 1;
}
```

$$\exists i \exists j : \Big(i < n \wedge j < n \wedge i \neq j \wedge \big(\underbrace{3i + 5 = 3j + 5}_{\text{zapis-zapis}} \vee \underbrace{3i + 5 = 3j + 7}_{\text{zapis-odczyt}} \vee \underbrace{3i + 7 = 3j + 5}_{\text{odczyt-zapis}} \big) \Big)$$

Formuła fałszywa dla każdego n, konfliktu nie ma, więc kompilator może zrównoleglić lub zwektoryzować (np. AVX-512).

Automatyczne dowodzenie twierdzeń

Poniższe procedury są pełnymi albo częściowymi solwerami dla arytmetyki Presburgera. Niektóre wspierają też proste heurystyki dla mnożenia czy potęgowania.

- Lean: `omega`
- Rocq: `lia`
- Isabelle/HOL: `presburger`

Literatura

- Presburger, M. (1929). *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt*
- Büchi, J. R. (1960). *Weak Second-Order Arithmetic and Finite Automata*
- Cooper, D. C. (1972). *Theorem-proving in arithmetic without multiplication*
- Bès, A. (2002). *A Survey of Arithmetical Definability*
- Haase, C. (2014). *Subclasses of Presburger Arithmetic and the weak EXP hierarchy*
- Pitchanathan, A., Ulmann, C., Weber, M., et al. (2021). *FPL: fast Presburger arithmetic through transprecision*
- Kliemann, F. (2018). *Deciding Presburger Arithmetic Using Automata Theory*
- Kern, T. (2023), *Regular Languages and Model Theory*