Seminar 2 - Exerciții rezolvate

Echipa AA

29 octombrie 2022

Cuprins

Notații

În acest document vom folosi următoarele notații:

- $\mathcal{R}$ - mulțimea tuturor mulțimilor recursive
- $\mathcal{RE}$ - mulțimea tuturor mulțimilor recursive-enumerabile
- $\mathcal{P}$ - mulțimea tuturor programelor.
- $\mathcal{I}$ - o mulțime arbitrară de intrări (sau, pe englezește, inputuri)

Vom nota descrierile problemelor, cu litere mari latine.
Predicatul corelat unei probleme îl vom scrie cu litere mici grecești indice numele problemei.
Programul corelate unei probleme îl vom scrie cu majusculă și litere latine indice numele problemei.
Mulțimea corelată unei probleme o vom scrie cu cu majusculă și litere ebraice indice numele problemei.

Recapitulare teorie

O problemă de decizie este o întrebare arbitrară pe o mulțime specifică de intrări la care se poate răspunde binar, "da" sau "nu", "1" sau "0".[1]

O să atașăm unei probleme de decizie, $PROB$, un predicat (o funcție), $\pi_{PROB} : \mathcal{I} \to \{0, 1\}$, astfel:

[1] Definirea formală a unei probleme se poate face în mai multe feluri, vezi Colorni și Tsoukiàs (2013) și Kroening și Strichman (2016), dar noi o să folosim abordarea din Giumale (2004).

$$\pi_{PROB}(i) = \begin{cases} 1 & \text{dacă răspunsul la problema } PROB \text{ pentru } i \text{ e "da"} \\ 0 & \text{dacă răspunsul la problema } PROB \text{ pentru } i \text{ e "nu"} \end{cases}$$

Notăm cu $\aleph_{PROB}$ mulțimea de adevăr a lui π_{PROB}:

$$\aleph_{PROB} = \{i \in \mathcal{I} \mid \pi_{PROB}(i) = 1\}$$

Atenție: Unui predicat s-ar putea să îi corespundă, sau nu, un program care oferă un răspuns binar pentru problema descrisă.

Exemplu, problema opririi (eng: Halting Problem)

> *PO*: Fiind dat un program arbitrat și o intrare oarecare pentru el, se va opri programul pe acea intrare?

Atașăm problemei opririi, PO, predicatul/funcția

$$\pi_{PO}((P, w)) = \begin{cases} 1 & P(w) \text{ se oprește} \\ 0 & \text{altfel} \end{cases}$$

Cu alte cuvinte predicatul/funcția primește la intrare un cuplu, format dintr-un program, $P \in \mathcal{P}$, și o intrare oarecare asociată programului P, intrarea $w \in \mathcal{I}$. Predicatul spune pentru acest cuplu un rezultat binar, care e răspunsul problemei opririi. Deoarece scrierea sub formă de tuplu (P, w) este anevoiasă, vom rescrie predicatul/funcția ca primind mai mulți parametrii când e nevoie.

Atenție: Pentru predicatul/funcția, π_{PO}, nu există un program (o funcție efectiv calculabilă), care să calculeze răspunsul la problema opririi, adică

$$\nexists Q \in \mathcal{P}, Q(P, w) = \begin{cases} 1 & P(w) \text{ se oprește} \\ 0 & \text{altfel} \end{cases}$$

Asociată predicatului π_{PO}, notăm cu $\aleph_{PO}$ mulțimea de adevăr a problemei opririi, PO.

$$\aleph_{PO} = \{(P, w) \in \mathcal{P} \times \mathcal{I} \mid \pi_{PO}(P, w) = 1\}$$

Reduceri Turing

Fie două predicate asociate la două probleme, $PROB_1$ și $PROB_2$:

$$\pi_{PROB_A} : \mathcal{I}_{\mathcal{A}} \to \{0, 1\}$$
$$\pi_{PROB_B} : \mathcal{I}_{\mathcal{B}} \to \{0, 1\}$$

Spunem că $PROB_A$ se reduce la $PROB_B$, dacă $\exists f : \mathcal{I}_A \to \mathcal{I}_B$, o funcție efectiv calculabilă[2] (și totală[3]), astfel încât:

$$\forall i_A \in \mathcal{I}_A, \pi_{PROB_A}(i_A) = 1 \iff \pi_{PROB_B}(\underbrace{f(i_A)}_{i_b}) = 1$$

și se notează așa[4]:

$$PROB_A \leq_T PROB_B$$

Atenție: Se mai practică notația:

$$\aleph_{PROB_A} \leq_T \aleph_{PROB_B}, \text{dacă}$$

$$\forall i_A \in \mathcal{I}_{\mathcal{A}}, i_A \in \aleph_{PROB_A} \iff \underbrace{f(i_A)}_{i_b} \in \aleph_{PROB_B}$$

[2] recursivă, vezi Post et al. (1944)

[3] Adică funcția este definită pentru toate elementele din domeniu

[4] Indicele "T" vine de la Turing. Cu toate acestea noi **NU** facem o reducere precum a definit-o el în teza lui de doctorat (vezi Turing (1939)). În cel mai strict sens noi facem o reducere prin mapare, de la mai multe la unu, notată în cărțile de specialitate $\leq_m$. Aceasta este mai specifică (vezi Soare (2016)), decât formularea generală introdusă de Post et al. (1944) a reducerii generale Turing.

Consecințe

$$\aleph \leq_T \beth$$

1. $\beth \in \mathcal{R} \Rightarrow \aleph \in \mathcal{R}$
2. $\beth \in \mathcal{RE} \Rightarrow \aleph \in \mathcal{RE}$
3. $\aleph \notin \mathcal{R} \Rightarrow \beth \notin \mathcal{R}$
4. $\aleph \notin \mathcal{RE} \Rightarrow \beth \notin \mathcal{RE}$

Exemplul 1 - VID (din seminar, rescris)

> VID: Are un program arbitrar proprietatea de a nu se opri pe nicio intrare?

Intuim[5] că problema VID nu nu este semidecidabilă, adică $\aleph_{VID} \notin \mathcal{RE}$.
Folosim consecința 4, deci vom reduce o problemă nici măcar semidecidabilă, $\overline{PO}$, la VID.

$$\overline{PO} \leq_T VID$$

Fie predicatele asociate acestor probleme:

$$\pi_{\overline{PO}}(P, w) = \begin{cases} 1 & P(w) \textbf{ nu} \text{ se oprește} \\ 0 & \text{altfel} \end{cases}$$

$$\pi_{VID}(Q) = \begin{cases} 1 & Q \text{ nu se oprește pe nicio intrare} \\ 0 & \text{altfel} \end{cases}$$

Atenție: Programele date ca argument acestor predicate, P și Q, nu au nicio legătură între ele. Sunt numele a două programe oarecare din $\mathcal{P}$.

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P}$$
$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{\overline{PO}}(P, w) = 1 \iff \pi_{VID}(\underbrace{f(P, w)}_{Q}) = 1$$

Adică funcția f[6] trebuie să construiască, pentru orice cuplu program-intrare care nu se oprește, un alt program, Q, care nu se oprește pe nicio intrare.

```
1 void Q(GenericInput v) { // își ignoră inputul
2   P(w);
3   return; // și așa în programul se oprea dacă trecea de apelul anterior
4 }
```

Demonstrăm dubla implicație pe rând

1. $\pi_{\overline{PO}}(P, w) = 1 \Rightarrow \pi_{VID}(\underbrace{f(P, w)}_{Q}) = 1$

   Dacă $P(w)$ nu se oprește $\Rightarrow Q$, nu se oprește pe nicio intrare.

2. $\pi_{VID}(\underbrace{f(P, w)}_{Q}) = 1 \Rightarrow \pi_{\overline{PO}}(P, w) = 1$

   Dacă Q nu s-a oprit pe nicio intrare, înseamnă că nu s-a ajuns niciodată la linia 3 din codul de mai sus $\Rightarrow P(w)$ nu se oprește.

Exercițiul 1 - PO_5

[5] S-ar putea ca intuiția să ne și înșele. Ea provine de la faptul că avem de verificat o proprietate pentru o infinitate de intrări și problema neopririi nu este semidecidabilă.

[6] Construirea unui alt program dat fiind un program inițial la intrare este o sarcină pe care compilatoarele o duc la capăt zilinc. Cu toate acestea ca să aratăm că funcția este efectiv calculabilă ar trebuie să folosim niște aspecte pe care le veți studia la materia "Limbaje Formale și Automate".

> PO_5: Se termină un program arbitrar pe intrarea 5?

Intuim că problema PO_5 este semidecidabilă, adică $\aleph_{PO_5} \in \mathcal{RE} \setminus \mathcal{R}$.

Pasul 1

Demonstrăm mai întâi, cu cunoștiințele din seminarul trecut, că $\aleph_{PO_5} \in \mathcal{RE}$. Scriem programul Sim_{PO_5}, astfel încât:

$$Sim_{PO_5}(Q) = \begin{cases} 1 & \text{dacă } Q(5) \text{ se termină} \\ \perp & \text{altfel} \end{cases}$$

```
1 bool Sim_PO_5(NumericProgram Q) { // NumericProgram - tipul unei funcții cu un input
  ↪  numeric
2   Q(5);
3   return true;
4 }
```

Pasul 2

Demonstrăm folosind consecința 3, deci vom reduce o problemă semidecidabilă, $PO \notin \mathcal{R}$, la PO_5.

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P}$$
$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{PO}(P, w) = 1 \iff \pi_{PO_5}(\underbrace{f(P, w)}_{Q}) = 1$$

Adică funcția f trebuie să construiască, pentru orice cuplu program-intrare care se oprește, un alt program, Q, care se oprește pe intrarea 5.

```
1 void Q(Number x) { // își ignoră inputul
2   P(w);
3 }
```

Demonstrăm dubla implicație pe rând

1. $\pi_{PO}(P, w) = 1 \Rightarrow \pi_{PO_5}(\underbrace{f(P, w)}_{Q}) = 1$

   Dacă $P(w)$ se oprește $\Rightarrow Q$, se oprește pe intrarea 5.

2. $\pi_{PO_5}(\underbrace{f(P, w)}_{Q}) = 1 \Rightarrow \pi_{PO}(P, w) = 1$

   Dacă Q se oprește pe 5, înseamnă că s-a ajuns la linia 3 din codul de mai sus $\Rightarrow P(w)$ se oprește.

Exercițiul 2 - $DOAR_5$

> $DOAR_5$: Se termină un program arbitrar **doar** pe intrarea 5?

Intuim că problema $DOAR_5$ nu este semidecidabilă, adică $\aleph_{DOAR_5} \notin \mathcal{RE}$. Fie predicatul ei:

$$\pi_{DOAR_5}(Q) = \begin{cases} 1 & Q \text{ se oprește doar pe intrarea 5} \\ 0 & \text{altfel} \end{cases}$$

Folosim consecința 4, deci vom reduce o problemă nici măcar semidecidabilă, $\overline{PO}$, la $DOAR_5$.

$$\overline{PO} \leq_T DOAR_5$$

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P}$$

$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{\overline{PO}}(P, w) = 1 \iff \pi_{DOAR_5}(\underbrace{f(P, w)}_{Q}) = 1$$

Adică funcția f trebuie să construiască, pentru orice cuplu program-intrare care nu se oprește, un alt program, Q, care se oprește pe doar pe 5.

```
1  void Q(Number w) {
2    if (w == 5)
3      return;
4    else
5      P(w);
6  }
```

Demonstrăm dubla implicație pe rând

1. $\pi_{\overline{PO}}(P, w) = 1 \Rightarrow \pi_{DOAR_5}(\underbrace{f(P, w)}_{Q}) = 1$

   Dacă $P(w)$ nu se oprește $\Rightarrow Q$ se va opri doar dacă intră pe prima ramură a `if`-ului, deci se va opri doar pe 5.

2. $\pi_{DOAR_5}(\underbrace{f(P, w)}_{Q}) = 1 \Rightarrow \pi_{\overline{PO}}(P, w) = 1$

   Dacă Q s-a oprit doar pe 5, s-a oprit doar prin apelul `return;` de la linia 3 $\Rightarrow P(w)$ nu se oprește.

Exercițiul 3 - EXP_2

> EXP_2: Calculează un program arbitrar 2^x pentru orice intrare $x \in \mathbb{N}$?

Intuim că problema EXP_2 nu este semidecidabilă, adică $\aleph_{EXP_2} \notin \mathcal{RE}$. Fie predicatul ei:

$$\pi_{EXP_2}(Q) = \begin{cases} 1 & \forall x \in \mathbb{N}, Q(x) = 2^x \\ 0 & \text{altfel} \end{cases}$$

Folosim consecința 4, deci vom reduce o problemă nici măcar semidecidabilă, $\overline{PO}$, la EXP_2.

$$\overline{PO} \leq_T EXP_2$$

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P}$$
$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{\overline{PO}}(P, w) = 1 \iff \pi_{EXP_2}(\underbrace{f(P, w)}_{Q}) = 1$$

Adică funcția f trebuie să construiască, pentru orice cuplu program-intrare care nu se oprește, un alt program, Q, astfel încât $\forall x \in \mathbb{N}, Q(x) = 2^x$.

```
1  // rulează funcția P asupra inputului x în steps pași
2  // dacă funcția a returnat în mai puțini pași, returnează valoarea ei
3  // altfel returnează o referință nulă
4  Boolean run_in_steps(BooleanFunction P, Input x, Integer steps);
5
6  Number Q(Number x) {
7    if (run_in_steps(P, w, x) != null)
8      while (1);
9    else
10     return pow(2, x);
11 }
```

Demonstrăm dubla implicație pe rând

1. $\pi_{\overline{PO}}(P, w) = 1 \Rightarrow \pi_{EXP_2}(\underbrace{f(P, w)}_{Q}) = 1$

   Dacă $P(w)$ nu se oprește $\Rightarrow \nexists x \in \mathbb{N}$ astfel încât Q să se oprească în x pași, deci se va ajunge mereu la linia 10.

2. $\pi_{EXP_2}(\underbrace{f(P, w)}_{Q}) = 1 \Rightarrow \pi_{\overline{PO}}(P, w) = 1$

   Dacă se ajunge mereu la linia 10 $\Rightarrow \forall x \in \mathbb{N}, P(w)$ nu se oprește în x pași, deci $P(w)$ nu se oprește.

Exercițiul 4 - $STOP_3$

> $STOP_3$: Se oprește un program arbitrar pe fix 3 intrari numerice?

Intuim că problema $STOP_3$ nu este semidecidabilă, adică $\aleph_{STOP_3} \notin \mathcal{RE}$. Fie predicatul ei:

$$\pi_{STOP_3}(Q) = \begin{cases} 1 & \text{Q se oprește pe fix 3 intrări} \\ 0 & \text{altfel} \end{cases}$$

Folosim consecința 4, deci vom reduce o problemă nici măcar semidecidabilă, $\overline{PO}$, la $STOP_3$.

$$\overline{PO} \leq_T STOP_3$$

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P}$$
$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{\overline{PO}}(P, w) = 1 \iff \pi_{STOP_3}(\underbrace{f(P, w)}_{Q}) = 1$$

Adică funcția f trebuie să construiască, pentru orice cuplu program-intrare care nu se oprește, un alt program, Q, care se oprește pe fix 3 intrări.

```
void Q(Number x) {
  if (0 <= x && x < 3)
    return;
  else
    P(w);
}
```

Demonstrăm dubla implicație pe rând

1. $\pi_{\overline{PO}}(P, w) = 1 \Rightarrow \pi_{STOP_3}(\underbrace{f(P, w)}_{Q}) = 1$

   Dacă $P(w)$ nu se oprește $\Rightarrow Q$, se va opri doar când intră pe prima ramură a `if`-ului.

2. $\pi_{STOP_3}(\underbrace{f(P, w)}_{Q}) = 1 \Rightarrow \pi_{\overline{PO}}(P, w) = 1$

   Dacă Q se oprește doar pe 3 intrări, înseamnă că se termină doar pe prima ramură a `if`-ului $\Rightarrow P(w)$ nu se oprește.

Exercițiul 5 - EQP

> EQP: Se opresc două programe arbitrare pe aceleași intrări? (programele primesc același tip de input)
> Reduceți atât PO, cât și VID la EQP.

Scriem predicatul logic asociat problemei:

$$\pi_{EQP}(Q_1, Q_2) = \begin{cases} 1 & \forall i \in \mathcal{I}, Q_1(i) \neq \perp \iff Q_2(i) \neq \perp \\ 0 & \text{altfel} \end{cases}$$

$VID \leq_T EQP$

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \to \mathcal{P} \times \mathcal{P}$$
$$\forall P \in \mathcal{P}, \quad \pi_{VID}(P) = 1 \iff \pi_{EQP}(\underbrace{f(P)}_{(Q_1,Q_2)}) = 1$$

Adică funcția f trebuie să construiască, pentru orice program care se oprește, un cuplu de două programe, (Q_1, Q_2) care se opresc sau nu în tandem.

```
void Loop(GenericInput v) { // își ignoră inputul
  while (1);
}

void Wrapper(GenericInput v) {
  P(w);
}
```

$$Q_1 = Loop, Q_2 = Wrapper$$

Demonstrăm dubla implicație pe rând

1. $\pi_{VID}(P) = 1 \Rightarrow \pi_{EXP}(\underbrace{f(P)}_{(Q_1,Q_2)}) = 1$

   Dacă P nu se oprește pe nicio intrare $\Rightarrow, Wrapper$ se oprește pe toate intrările pe care se oprește și $Loop$.

2. $\pi_{EXP}(\underbrace{f(P)}_{(Q_1,Q_2)}) = 1 \Rightarrow \pi_{VID}(P) = 1$

   Dacă $Wrapper$ se oprește pe toate intările pe care se oprește și $Loop \Rightarrow P$ nu se oprește pe nicio intrare.

$PO \leq_T EQP$

Pentru a face reducerea trebuie să găsim o funcție de mapare efectiv calculabilă, care satisface constrângerile:

$$f : \mathcal{P} \times \mathcal{I} \to \mathcal{P} \times \mathcal{P}$$
$$\forall (P, w) \in \mathcal{P} \times \mathcal{I}, \quad \pi_{PO}(P, w) = 1 \iff \pi_{EQP}(\underbrace{f(P, w)}_{(Q_1,Q_2)}) = 1$$

Adică funcția f trebuie să construiască, pentru orice cuplu program-intrare care se oprește, un cuplu de două programe, (Q_1, Q_2) care se opresc sau nu în tandem.

```
void All(GenericInput v) { // își ignoră inputul
  return; // se oprește mereu
}

void Wrapper(GenericInput v) {
  P(w);
}
```

$$Q_1 = All, Q_2 = Wrapper$$

Demonstrăm dubla implicație pe rând

1. $\pi_{PO}(P, w) = 1 \Rightarrow \pi_{EXP}(\underbrace{f(P, w)}_{(Q_1,Q_2)}) = 1$

   Dacă $P(w)$ se oprește $\Rightarrow Wrapper$ se oprește pe aceleași intrări ca și All.

2. $\pi_{EXP}(\underbrace{f(P, w)}_{(Q_1,Q_2)}) = 1 \Rightarrow \pi_{PO}(P, w) = 1$

   Dacă $Wrapper$ se oprește pe toate intrările pe care se oprește și $All \Rightarrow P(w)$ se oprește.

Bibliografie

Colorni, Alberto, și Alexis Tsoukiàs. 2013. „What is a decision problem? Preliminary statements". În *International Conference on Algorithmic Decision Theory*, 139–53. Springer.

Giumale, Cristian A. 2004. „Introducere în ANALIZA ALGORITMILOR". *Editura Polirom, Bucuresti*.

Kroening, Daniel, și Ofer Strichman. 2016. *Decision procedures*. Springer.

Post, Emil L et al. 1944. *Recursively enumerable sets of positive integers and their decision problems*. World Scientific. https://projecteuclid.org/journals/bulletin-of-the-american-mathematical-society/volume-50/issue-5/Recursively-enumerable-sets-of-positive-integers-and-their-decision-problems/bams/1183505800.pdf.

Soare, Robert I. 2016. *Turing computability: Theory and applications*. Springer.

Turing, Alan Mathison. 1939. „Systems of logic based on ordinals". *Proceedings of the London Mathematical Society, Series 2* 45: 161–228. https://www.dcc.fc.up.pt/~acm/turing-phd.pdf.