Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Absztrakt interpretációt használó keresési stratégiák Petri-háló alapú modellekhez

TDK DOLGOZAT

Készítette

Farkas Rebeka

Konzulensek

Vörös András, tudományos segédmunkatárs
Tóth Tamás, doktorandusz
Semeráth Oszkár, doktorandusz

2014. október 22.

Kivonat

A Petri-hálók az aszinkron, elosztott, konkurens, párhuzamos és nemdeterminisztikus rendszerek elterjedt grafikus és matematikai modellező eszközei. Petri-háló alapú modellek analízisekor fontos kérdés az állapotelérhetőség, amely során azt vizsgáljuk, hogy egy adott célállapot része-e az állapottérnek. A probléma egyszerű Petri-hálók esetén eldönthető, azonban a végtelen állapottér és a Petri-hálók nagy kifejezőereje miatt a nagyon komplex, az EXPSPACE-nehéz komplexitási osztályba tartozik. A tiltó-élekkel kibővített Petri-hálók kifejezőereje azonos a Turing gépekkel, emiatt az elérhetőségi probléma ebben az esetben már nem eldönthető.

Végtelen állapotterű modellek analízisére nyújtanak megoldást a különböző, matematikai absztrakción alapuló módszerek. Ezen módszerek előnye, hogy egy véges állapottér reprezentációt építve és azt bejárva vizsgálják az elérhetőségi és/vagy invariáns tulajdonságokat. Az absztrakt interpretáció algoritmusa a véges reprezentációt konvex poliéderek segítségével állítja elő, így felülbecsülve a lehetséges elérhető állapothalmazt. Petri-háló alapú modellek esetén azonban az irodalomban ismert módszerek sokszor nem képesek elég precíz becslést adni. Munkám során ezen módszerek továbbfejlesztésével és kiterjesztésével foglalkoztam.

A dolgozatban bemutatok olyan, az absztrakt interpretáció finomításán alapuló algoritmusokat, amelyek segítségével olyan rendszerek vizsgálhatóságát is lehetővé tettem, amelyre a korábbi algoritmusok nem voltak képesek. Kifejlesztettem egy új algoritmust, amely a felderített állapotteret finomítja az elérhetőségi feltétel figyelembevételével. Emellett pedig mutatok egy módszert, amely az explicit állapottér bejáró és az absztrakt interpretáción alapuló algoritmusokat együttesen alkalmazza a hatékonyság növelése érdekében. Az új algoritmusok előnyeit mintapéldákkal szemléltetem.

Abstract

Petri nets are a common graphical and mathematical modelling tool for asynchronous, distributed, concurrent, parallel and non-deterministic systems. When analysing Petri net based models, reachability - i.e. whether or not the state space includes a given target state - becomes an important question. The problem is decidable for simple Petri nets, however, due to the infinite state space and the expressive power of Petri nets, it is EXPSPACE-hard. The expressive power of Petri nets extended by inhibitor arcs is equivalent to that of Turing machines, so in their case the reachability problem remains undecidable.

Several methods based on mathematical abstraction provide a solution for the analysis of models with infinite state space. The advantage of these methods is that they examine reachability and/or invariant properties by building and exploring a finite representation of the state space. The abstract interpretation algorithm produces the finite representation using convex polyhedra, thus overapproximating the set of reachable states. However, known methods for Petri net based models are often not able to give a sufficiently precise approximation. In my work I dealt with further development and extension of these methods.

In the report I present methods based on refining abstract interpretation which allow the examination of systems that previous algorithms weren't able to handle. I developed a new algorithm that refines the explored state space by taking the reachability condition into account. I also present a method that combines abstract interpretation based solutions with algorithms that explicitly explore the state space to increase efficiency. I demonstrate the advantages of the new algorithms by examples.

Tartalomjegyzék

1. fejezet

Bevezető

Model allapú rendszertervezés során fontos meggyőződnünk a tervek, azaz a rendszer modelljének helyességéről. Ebben segítenek nekünk a formális modellek és formális analízis technikák.

A Petri-hálók az aszinkron, elosztott, konkurens, párhuzamos és nemdeterminisztikus rendszerek elterjedt grafikus és matematikai modellező eszközei. Formális analízis során, így a Petri-háló alapú modellek analízisekor is fontos kérdés az állapotelérhetőség, amely során azt vizsgáljuk, hogy egy adott célállapot része-e az állapottérnek, azaz elérhető-e a kezdőállapotból engedélyezett átmenetek végrehajtásával. A probléma egyszerű Petri-hálók esetén eldönthető, azonban a végtelen állapottér és a Petri-hálók nagy kifejezőereje miatt a nagyon komplex, az EXPSPACE-nehéz komplexitási osztályba tartozik. A tiltó-élekkel kibővített Petri-hálók kifejezőereje azonos a Turing gépekkel, emiatt az elérhetőségi probléma ebben az esetben már algoritmikusan nem eldönthető.

Végtelen állapotterű modellek analízisére nyújtanak megoldást a különböző, matematikai absztrakción alapuló módszerek. Ezen módszerek előnye, hogy egy véges állapottér reprezentációt építve és azt bejárva vizsgálják az elérhetőségi és/vagy invariáns tulajdonságokat. Az absztrakt interpretáció algoritmusa a véges reprezentációt konvex poliéderek segítségével állítja elő, így felülbecsülve a lehetséges elérhető állapothalmazt. Petri-háló alapú modellek esetén azonban az irodalomban ismert módszerek sokszor nem képesek elég precíz becslést adni. Munkám során ezen módszerek elméleti áttekintésével, továbbfejlesztésével és kiterjesztésével foglalkoztam.

A dolgozatban bemutatok olyan, az absztrakt interpretáció finomításán alapuló algoritmusokat, amelyek segítségével új típusú rendszerek vizsgálhatóságát is lehetővé tettem, amelyre a korábbi algoritmusok nem voltak képesek. Kifejlesztettem egy új algoritmust, amely a felderített állapotteret finomítja az elérhetőségi feltétel figyelembevételével. Emellett pedig mutatok egy módszert, amely az explicit állapottér bejáró

és az absztrakt interpretáción alapuló algoritmusokat együttesen alkalmazza a hatékonyság növelése érdekében. Az új algoritmusok előnyeit mintapéldákkal szemléltetem.

A dolgozat felépítése a következő. A 2. fejezetben a dolgozat megértéséhez szükséges háttérismereteket mutatom be. A 3. fejezetben bemutatok egy új módszert, amely figyelembe veszi az elérhetőségi kérdést. A 4. fejezetben további absztrakciófinomításra alkalmas megközelítéseket ismertetek. Az 5. fejezetben leírom, hogyan lehet absztrakt interpretáció segítségével lineáris tranzíciós rendszereken elérhetőségi analízist végezni, valamint ismertetem a bemuatott algoritmusok implementációs lehetőségeit.

2. fejezet

Háttérismeretek

Ebben a fejezetben áttekintem a dolgozat további részéhez szükséges ismereteket és definíciókat. Bemutatok több Petri-háló alapú viselkedésmodellt, és a velük kapcsolatos fogalmakat, egy általános struktúrát ezen modellek leírására, ismeretetem az absztrakt interpretáció alapjait, valamint bemutatom a munkám alapját képző algoritmust.

2.1. Petri-háló alapú viselkedésmodellek

A Petri-hálók az aszinkron, elosztott, konkurens, párhuzamos és nemdeterminisztikus rendszerek széles körben alkalmazott grafikus és matematikai modellező eszközei. Kifejezőerejük növelésének érdekében számos kiegészítést definiáltak, emellett az évek során többféle, a Petri-hálókkal számításelméleti szempontból (kifejezőerőben) ekvivalens, ám modelltervezési szempontból (leíróerőben) bővebb funkciókkal rendelkező viselkedésmodell látott napvilágot. A dolgozatban ezek közül a Vektor-alapú Számláló Rendszerek (Vector Addition System), azok vezérlési állapotokkal kiterjesztett változatával, illetve tiltó élekket tartalmazó Petri-hálókkal foglalkozom.

2.1.1. Egyszerű Petri-háló

Az Petri-háló formális definíciója a következő.

Definíció 1 (Petri-háló) *A Petri-háló egy* $PN = (P, T, E, W, M_0)$ *struktúra, ahol:*
$P = \{p_1, p_2, \ldots, p_k\}$ *a helyek véges halmaza,*
$T = \{t_1, t_2, \ldots, t_n\}$ *a tranzíciók véges halmaza,*
$E \subseteq (P \times T) \cup (T \times P)$ *az élek halmaza,*
$W : E \mapsto \mathbb{Z}^+$ *a súlyfüggvény,*
$M_0 : P \mapsto \mathbb{N}$ *a kezdeti tokeneloszlás.*

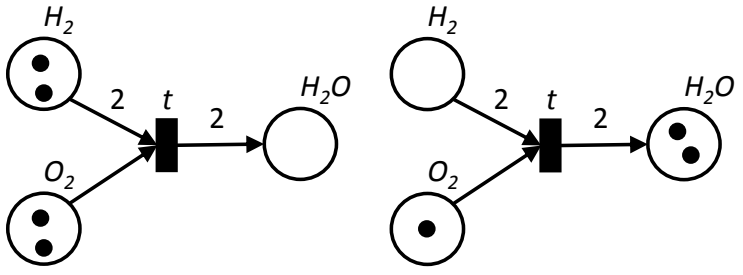
Szavakkal ez úgy fogalmazható meg, hogy az egyszerű Petri-háló *helyek*, *tranzíciók*, a helyekhez rendelt *tokenek*, és helyek és tranzíciók között vezető *élek* halmaza. Ez utóbbi két részhalmaz uniója - az egyik a helyről tranzícióba, a másik tranzícióból helyre vezető élek halmaza.

Grafikusan irányított, súlyozott páros gráfként reprezentálható, ahol a helyeket körök, a tranzíciókat téglalapok jelölik, az élek pedig irányítottak, a megfelelő *élsúly* feltüntetésével. A dolgozatban p helyről t tranzícióba vezető él súlyát $w_{in}(p,t)$, a t-ből p-be menő élét $w_{out}(t,p)$ írja le. A tokeneket a helyekre (körökbe) rajzolt pontok jelölik. Ezek száma, a *tokeneloszlás* határozza meg a Petri-háló állapotát.

Állapotátmenet akkor következik be, ha egy engedélyezett tranzíció tüzel. A tüzelési szabályok a következők:

- Egy t tranzíció *engedélyezett*, ha t minden p bemenő helyén van legalább $w_{in}(p,t)$ token.
- Egy engedélyezet tranzíció tüzelhet, de nem feltétlenül tüzel. Több tranzíció esetén bármelyik tüzelhet,tehát a Petri-háló viselkedése nemdeterminisztikus.
- Ha egy engedélyezett t tranzíció tüzel, t minden p_i bemeneti helyéről elvesz $w_{in}(p_i,t)$ tokent, és minden p_o kimeneti helyén elhelyez $w_{out}(t,p_o)$ tokent.

Példa 1 (Tranzíciótüzelés) *A 2.1. ábrán egy Petri-háló látható, amely egy egyszerű kémiai folyamatot modellez [10]. A hálóban egy tranzíció (t) és három hely (H_2, O_2, H_2O) található. A 2.1a. ábrán a t tranzíció engedélyezett. A tranzíció tüzelésével a 2.1a. ábrán látható állapotból a 2.1b. ábrán látható állapotba kerül a háló. Ebben az állapotban nincs engedélyezett tranzíció.*



(a) Kezdő tokeneloszlás (b) Tokeneloszlás t tüzelése után

2.1. ábra. Petri-háló állapotátmenete

Elérhetőségi probléma

A Petri-háló állapotát tehát a tokeneloszlása határozza meg. Egy engedélyezett tranzíció tüzelése megváltoztatja a tokeneloszlást. A dolgozatban $M \xrightarrow{t} M'$ jelöli, ha a rendszer M állapotból t eltüzelésének hatására M' állapotba kerül.

Egy M_n állapot *elérhető* M_0 kezdőállapotból, ha létezik engedélyzett tüzelési sorozat M_0-ból, aminek hatására M_n lesz a tokeneloszlás. A Petri-háló összes elérhető állapotának halmazát $R(PN, M_0)$, vagy egyszerűbben $R(M_0)$ jelöli. Az *elérhetőségi probléma* annak az eldöntése, hogy egy adott M_n tokeneloszlás (célállapot) elérhető-e M_0 kezdőállapotból, azaz igaz-e, hogy $M_n \in R(M_0)$. A Petri-hálók elérhetőségi problémája bizonyítottan eldönthető [9], azonban az EXPSPACE-nehéz komplexitási osztályba tartozik [8], azaz nem létezhet olyan algoritmus, amely minden esetben hatékonyan oldja meg a problémát.

A gyakorlatban sokszor előfordul, hogy nincs konkrét megadott célállapot, csak az érdekel, hogy elérhető-e olyan állapot, amire teljesülnek bizonyos feltételek. Ekkor beszélünk *rész-elérhetőségről*. Ilyenkor leggyakrabban a kívánt eredmény a *nem* válasz, mivel egy rendszer biztonságossága azt jelenti, hogy nem juthat el nem kívánt állapotba. Ha a kedvezőtlen állapotokra megfogalmazott feltételekkel leírt állapotokról belátjuk, hogy nem elérhetőek, akkor a rendszer biztonságos.

Elérhetetlenség bizonyítására kézenfekvőnek tűnő módszer bejárni az egész állapotteret, majd ellenőrizni, hogy a kedvezőtlen állapot része-e. Ehhez állapotgráfot kell építeni a következő módon.

A kezdőállapotból, mint az állapotgráf gyökeréből kiindulva minden engedélyezett tranzíciót eltüzelünk, majd minden kapott állapothoz felveszünk egy gráfcsúcsot, amelybe a kezdőállapotból vezetünk egy élet, megjelölve, hogy mely tranzíció hatására érünk el az állapotba. A kapott új állapotokkal, mint kezdőállapottal megtesszük ugyanezt. Ha egy tüzelés hatására olyan állapotot kapunk, amivel már találkoztunk, akkor az élet a megfelelő csúcsba húzzuk. Ezt addig folytatjuk, míg új állapotokat kapunk. Ekkor a gráfból kiolvasható az állapottér, és az elérési utak.

A módszer egyszerű, azonban a veszélyes állapotok halmaza, ahogy állapottér is könnyen lehet végtelen, így ez a megoldás nem kivitelezhető. A gyakorlatban az elérhetetlenség (vagy akár rész-elérhetetlenség) bizonyítása invariánsokkal történik.

Definíció 2 (Invariáns) *Adott Ψ tranzíciós rendszerre Φ kényszer invariáns, ha Ψ minden elérhető állapotában teljeseül Φ.*

Invariáns tulajdonságok kereséséhez nem kell bejárni az egész állapotteret, annak teljesülése sokszor más módon is bizonyítható. Amennyiben találunk egy olyan invariánst, ami kizárja a kedvezőtlen állapotokat, akkor ezek az állapotok biztosan nem elérhetők.

2.1.2. Tiltó élekkel kiegészített Petri-háló

A Petri-hálóknak számos módosított változata létezik. Egyik legnépszerűbb a tiltó élekkel való kiterjesztés, amely a Petri-hálók kifejezőerejét a Turing-gépekével azonos szintre emeli [1]. A tiltó élekkel kiegészített Petri-háló formális definíciója a következő:

Definíció 3 (Tiltó élekkel kiegészített Petri-háló) *A tiltó élekkel kiegészített Petri-háló egy* $PN_I = (PN, I, WI)$ *strukrúra, ahol:*
$PN = (P, T, F, W, M_0)$ *egy Petri-háló,*
$I \subseteq (P \times T)$ *a tiltó élek halmaza,*
$W_I : I \to \mathbb{Z}_+$ *a tiltó élekhez tartozó súlyfüggvény.*

Szavakkal ez azt jelenti, hogy az eredeti Petri-háló struktúra kiegészül helyekről tranzíciókba mutató súlyozott tiltó élekkel. A grafikus reprezentációban a tiltó élek végén nyíl helyett egy üres kör szerepel. A dolgozatban a p helyről t tranzícióba vezető tiltó él súlyát $w_i(p, t)$ jelöli.

A tiltó élekkel való kiterjesztés az engedélyezettségi szabály megváltoztatásával, pontosabban egy új engedélyezettségi feltétel bevezetésével módosítja a Petri-háló viselkedését, és ezáltal szűkíti az elérhető állapotok halmazát (állapotteret). A módosított feltétel szerint egy t tranzíció engedélyezett, ha

- t minden p bemenő helyén legalább $win(p, t)$ token található, *ÉS*
- minden olyan p_i bemenő helyén, ahonnan vezet tiltó él kevesebb, mint $w_i(p_i, t)$ token található.

2.1.3. Vektor-alapú Számláló Rendszerek

A Vektor-alapú Számláló Rendszerek (*Vector Addition System*, VAS) formális definiíciója a következő:

Definíció 4 (Vektor-alapú Számláló Rendszer)
A Vektor-alapú Számláló Rendszer egy $k \in \mathbb{Z}_+$ *dimenziós* $VAS = (v, A)$ *pár, ahol*

- $v \in \mathbb{N}^k$ *a* kezdővektor*, és*
- $A \subseteq \mathbb{Z}^k$ *az összeadandók halmaza.*

A Vektor-alapú Számláló Rendszerekre felírt elérhetőségi problémánál azt vizsgáljuk, hogy van-e olyan $v_0 \dots v_n$ sorozat, ahol v_0 a kezdővektor, v_n a célvektor és a sorozat minden $v_i \in \mathbb{N}^k$ elemére $v_{i+1} - v_i \in A$.

A Vektor-alapú Számláló Rendszereknek létezik egy kiterjesztése (*Vector Addition System with States*, VASS), ami a Vektor-alapú Számláló Rendszer struktúráját kiegészíti *vezérlési állapotokkal* (*state*, tranzíciós rendszereknél

a *location* megfelelője). Grafikusan irányított gráfként reprezentálható, ahol a csúcsok a Vektor-alapú Számláló Rendszer vezérlési állapotait, az élek pedig az A halmaz elemeit jelölik. Az elérhetőségi problémánál azt vizsgáljuk, hogy van-e olyan út a gráfban, ahol v kezdővektorból, a kijelölt kezdeti vezérlési állapotból indulva, az érintett élekre írt vektorok hozzáadásával a kérdéses célvektort kapjuk.

A Vektor-alapú Számláló Rendszereket Karp és Miller vezette be [7], számításelméleti ekvivalenciájukat az egyszerű (tiltó él nélküli) Petri-hálókkal Hack bizonyította [4]. A vezérlési állapotokkal kiterjesztett Vektor-alapú Számláló Rendszer bevezetése, valamint ekvivalenciájának bizonyítása a fenti struktúrákkal Hopcroft és Pansiot 1979-ben született cikkéhez [6] fűződik.

Az ekvivalencia szemléltetéséhez a 2.2. ábrán bemutatom a [6] cikkben definiált rendszer különböző struktúrákkal megadott viselkedésmodelljét.

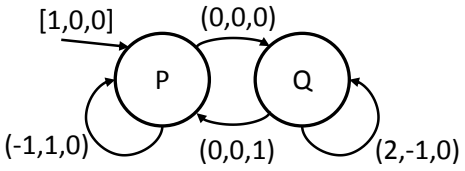
2.2. Lineáris tranzíciós rendszerek

A lineáris tranzíciós rendszerek formális definíciója [12] alapján a következő.

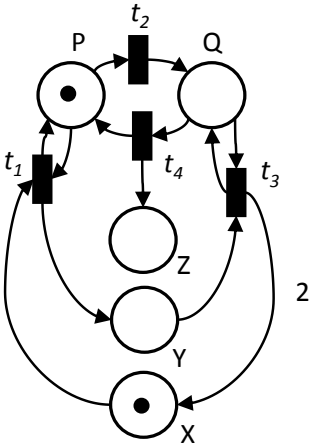
Definíció 5 (Lineáris tranzíciós rendszerek) *A* lineáris tranzíciós rendszer *egy* $LTS = (L, T, l_0, \Theta)$ *struktúra a változók egy* V *halmaza felett, ahol*

- L *a vezérlési állapotok halmaza;*
- T *a tranzíciók halmaza, ahol minden* $\tau \in T$ *tranzíció* $\tau = (l_i, l_j, \rho_\tau)$ *alakú struktúra, ahol* $l_i, l_j \in L$ *azon vezérlési állapotokat jelölik, ahol a rendszer a tranzíció tüzelését megelőzően tartózkodott, illetve ahová a tüzelés hatására került;* ρ *pedig egy lineáris kényszer(halmaz)* $V \cup V'$ *felett, ahol* V *jelöli a változók értékét a rendszer tüzelés előtti,* V' *pedig a tüzelés utáni állapotában;*
- $l_0 \in L$ *a kiindulási vezérlési állapot;*
- Θ *pedig a kezdeti kikötéseket leíró kényszer* V *felett.*

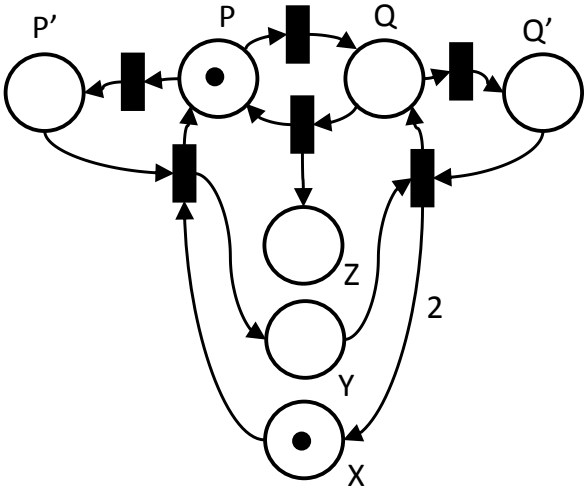
A fenti definíció az eddig ismeretett struktúrák általánosítása, illetve kiterjesztése. A változók és a vezérlési állapotok a Vektor-alapú Számláló Rendszereknél látott struktúrának felelnek meg, bár itt már akár valós változókról is beszélhetünk. A tranzíciókra felírt lineáris kényszerhalmaz a Petri-hálóknál látott engedélyezési feltétel, illetve hatás, azaz a tranzíciós rendszereknél ismert őrfeltétel illetve akció általános leírása. Megjegyzendő, hogy itt egy változó állapota nem feltétlenül csak a saját előző állapotától függ, felírható tetszőleges lineáris értékadás pl. $x' = x + y + 1$, ezen kívül ez a modell kezdőállapot helyett kezdőállapothalmazból indul ki.



(a) Eredeti, vezérlési állapotokkal kiterjesztett Vektor-alapú Számláló Rendszer alak



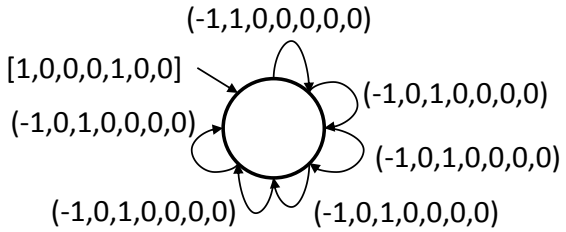
(b) Petri-háló alak



(c) Hurokél nélküli Petri-háló alak

```
v(1,0,0,0,1,0,0)
A{(-1,1,0,0,0,0,0),
  (-1,0,1,0,0,0,0),
  (1,0,-1,0,0,0,1),
  (0,0,-1,1,0,0,0),
  (1,-1,0,0,-1,1,0),
  (0,0,1,-1,2,-1,0)}
```

(d) Vektor-alapú Számláló Rendszer alak



(e) Vektor-alapú Számláló Rendszer alak grafikus megjelenítése

2.2. ábra. Egy rendszerhez tartozó viselkedésmodellek

Példa 2 (Lineáris tranzíciós rendszer) *Példaként a 2.2a. ábrán látható Vektor-alapú Számláló Rendszer lineáris tranzíciós rendszerként a következő:*
$V = \{x, y, z\}$
$L = \{P, Q\}$
$T = \{$
$t_1(P, P, \{x \geq 1 \wedge x' = x - 1 \wedge y' = y + 1\}),$
$t_2(P, Q, \emptyset),$
$t_3(Q, Q, \{y \geq 1 \wedge x' = x + 2 \wedge y' = y - 1\}),$
$t_4(Q, P, z' = z + 1)\}$
$l_0 = P$
$\Theta = \{x = 1 \wedge y = 0 \wedge z = 0\}$

2.3. Absztrakt interpretációt használó ellenőrzési módszerek

Az absztrakt interpretáció célja, hogy invariánsok segítségével alátámassza bizonyos kedvezőtlen állapotok elérhetetlenségét. Ehhez először felépít egy felülbecslüt (absztrakt) állapottér-reprezentációt, majd ellenőrzi, hogy ennek része-e a kérdéses állapothalmaz. Ha az absztrakt állapottér-reprezentáció nem tartalmaz egy állapotot, akkor az a valódi állapottérnek sem része, tehát az állapot nem elérhető. Fordítva ez nem teljesül, tehát előfordulnak bizonyos állapotok, amelyek nem részei a valódi állapottérnek, de az absztrakt reprezentációnak igen, ezért elérhetőséget nem lehet absztrakciós módszerekkel alátámasztani. Ha a kérdéses állapotról (állapothalmazról) nem sikerült bizonyítani, hogy nem elérhető, érdemes megpróbálkozni az absztrakció finomításával.

2.3.1. Poliéder, mint felsőbecslés

Az absztrakt interpretáció során tehát az alkalmazott felső becslés (*abstract domain*) kulcskérdés, mivel elég precíznek kell lennie ahhoz, hogy minél több állapot elérhetetlenségét bizonyítani lehessen vele, viszont az algoritmus hatékonyságához az is szükséges, hogy egyszerűen reprezentálható és az állapotátmenet kiszámítható legyen. Én egy olyan módszert vizsgáltam, ahol a felső becslést többdimenziós konvex poliéderek segítségével végezték.

Konvex poliéderek

Definíció 6 (Konvex poliéder) *Konvex poliédernek nevezzük azon térbeli ponthalmazokat, melyek előállnak lineáris egyenletrendszerek megoldáshalmazaként.*

Egy konvex poliéder egyértelműen meghatároz egy lineáris egyenlőtlenségrendszert. Ezen egyenlőtlenségrendszer elemei lesznek az ellenőrzés illetve a bizonyítás során használt invariánsok. Konvex poliéderek tehát lineáris kényszerek (lineáris invariánsok) keresésére alkalmazhatók.

Bármely véges dimenziószámú vektortérben megadható konvex poliéder a fenti módon, tetszőleges lineáris egyenlőtlenségrendszerrel. Ebből adódóan lehetséges olyan konvex poliéder definiálása, amely nincs minden irányból határolva, azaz nem korlátos.

Poliéderek és Petri-hálók állapotterének kapcsolata

Az előző részekben láthattuk, hogy a Petri-háló állapota leírható egy egészvektorral, amely dimenzióinak száma megegyezik a háló helyeinek a számával. A vektor koordinátái az adott helyen abban az állapotban lévő tokenek száma. Ezek alapján a Petri-háló minden egyes állapota megfeleltethető egy pontnak a sokdimenziós térben. A Petri-háló állapottere tehát egy ponthalmaz a térben, amelyet az absztrakt interpretáció során felülről közelítünk egy konvex poliéderrel. Fontos megjegyezni, hogy az absztrakt állapottér-reprezentáció nem feltétlenül a legkisebb olyan poliéder, amely tartalmazza az állapottér összes pontját, mivel ennek megtalálása meglehetősen nehéz, vagy akár lehetetlen feladat.

A Petri-háló viselkedése jól közelíthető poliédereken végezhető műveletek segítségével. A tranzíciók engedélyezési feltételei lineáris egyenlőtlenségek, így minden tranzícióra megadható egy konvex poliéder, mely pontosan azon állapotoknak megfeleltetett rácspontokat tartalmazza, amelyekben a tranzíció engedélyezett. Ezáltal egy adott állapottér-reprezentáció állapotai közül egyszerűen kiválaszthatóak azok, ahol egy tranzíció engedélyezve van – venni kell az állapottér-reprezentáció és a tranzíció engedélyezési poliéderek metszetét (szintén konvex poliéder). Az engedélyezett helyek kiválasztása után a tranzíció eltüzelése nem más, mint egy térbeli eltolás, amelyhez az eltolásvektort a tranzíció hatása határozza meg.

Példa 3 (Tranzíciók és poliéder műveletek kapcsolata) *Vegyük a 2.1. ábrán látható Petri-hálót. Kiindulási állapota a* $(2;2;0)$ *térbeli koordinátákkal leírt állapotoknak feleltethető meg. A tranzíció engedélyezési feltétele, hogy* H_2 *helyen legalább 2,* O_2 *helyen legalább 1 token, tehát a tranzícióhoz tartozó engedélyezési poliéder a* $(H_2 \geq 2) \wedge (O_2 \geq 1)$ *lineáris egyenletrenszerrel írható le. Tüzeléskor* H_2 *helyről 2,* O_2 *helyről 1 token tűnik el,* H_2O *helyen pedig megjelenik egy. Az ezt leíró vektor a* $(-2;-1;1)$ *koordinátákkal írható fel.*

$$(2;2;0) \cap ((x \geq 2) \wedge (y \geq 1)) = (2;2;0)$$

$$(2;2;0) + (-2;-1;1) = (0;1;1)$$

A tranzíció eltüzelése után tehát a tokeneloszlás $(0;1;1)$*, azaz* H_2 *helyen 0,* O_2 *helyen 1, és* H_2O *helyen is 1 token található.*

Az elérhetőség ellenőrzése során felhasználható az a tudás, hogy bár a poliéder tartalmaz minden pontot, amire igazak a lineáris egyenlőtlenségekkel leírt tulajdonságok, a Petri-háló lehetséges állapotait csak pozitív koordinátájú rácspontok jelölhetik.

2.3.2. Algoritmus

Petri-hálókon végzett elérhetőségi analízishez Clarisó, Rodríguez-Carbonell, és Cortadella definiálta az absztrakt interpretációs algoritmust[2]. Ebben a részben ezt muatom be – először az algoritmus pszeudokódját ismertetem, majd a felhasznált wideing operátort, végül egy példa segítségével elemzem a lefutását.

Pszeudokód

Az algoritmus, ahogy azt Clarisó, Rodríguez-Carbonell, és Cortadella leírta:

Input: $PN = (P, T, E, W, M_0)$ Petri-háló
Output: $R(PN, M_0)$-ra adott felső becslés (itt: a poliédert leíró invariánsok)

```
function BASIC(PN)
    reachable = {M_0}
    repeat
        old := reachable
        for all transition t ∈ T do
            enabling:=enablingCondition(t)
            enabled := reachable ∩ enabling
            if enabled = ∅ then continue
            end if
            next := fire(t, enabled)
            reachable := reachable ▽ ( reachable ∪ next )
        end for
    until reachable = old
     return reachable
end function
```

Az érthetőség kedvéért szavakkal megfogalmazva a következő zajlik le:

1. Kiindulunk a kezdőállapotból.

2. Eltüzelünk egy engedélyezett tranzíciót.

3. Az állapotteret kiterjesztjük a kapott új állapottal.

4. Veszünk egy tranzíciót. Megnézzük, hogy a jelenlegi állapottér reprezentáción belül hol engedélyezett, majd itt eltüzeljük. (Poliéder eltolás segítségével ez az összes engedélyezett állapotból egyszerre, egyetlen művelettel tehető meg.)

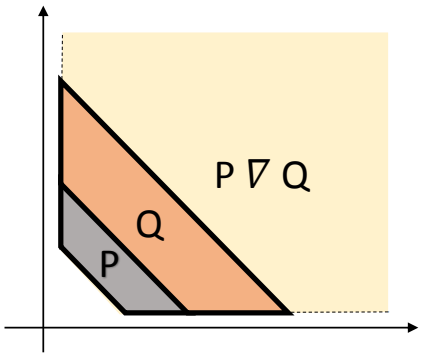
5. Az állapotteret kiterjesztjük a kapott új állapotokkal.

6. Ezt folytatjuk egészen addig, amíg új állapotokkal tudjuk bővíteni a reprezntációt. Ekkor véget ér az algoritmus.

A widening operátor

Az algoritmus során az új állapotok felvétele „veszélyes" lépés, ugyanis az adott poliéderrel vett konvex unió még nem lenne elég ahhoz, hogy az algoritmus terminálódni tudjon, hiszen végtelen állapotteret csak nem korlátos poliéderrel lehet reprezentálni, az új állapottal vett konvex unió viszont minden esetben korlátos poliédert ad eredményül. Ennek a problémának a megoldására szolgál a *widening operátor* bevezetése.

Számos, különböző struktúrákra értelmezett widening operátor létezik. A pontos definíció [5] nélkülözésével az algoritmus során konvex poliéderekre alkalmazott widening operátor működése egyszerűen leírható: $P \bigtriangledown (P \cup Q)$ az a poliéder amelyet a P-t leíró lineáris kényszerek közül a Q-ra nem teljesülőek elhagyásával kapott lineáris egyenlőtlenségrendszer meghatároz. A widening operátorok formális definíciójában leírt követelmény teljesüléséből következik az algoritmus terminálódása. Az érthetőség kedvéért a widening operátor működését a 2.3. ábra szemlélteti.



2.3. ábra. Poliéder-kiterjesztés widening operátor segítségével

Példa

Példa 4 (Az algoritmus futása) *Példaként először vegyük a 2.4. ábrán látható egyszerű termelő-fogyasztó rendszert. Az ábrán t_1 a termelőt, t_3 pedig a fogyasztót jelképezi, p_2 pedig a t_2 miatt veszteséges csatorna. A működés során a t_1 által termelt mennyiség p_1 számlálón, a t_3 által fogyasztott mennyiség p_3 számlálón jelenik meg. Az állapottér-reprezentáció alkaulását a 2.5. ábrán szemléltetem.*

Az algoritmus elindul $M_0 = (0, 0, 0)$ kezdőállapotból, eltüzeli az egyetlen engedélyezett tranzíciót (t_1), melynek hatására a kapott tokeneloszlás $M_1 = (1, 1, 0)$. A két pont konvex uniója a

$$(p_3 = 0) \wedge (p_1 - p_2 = 0) \wedge (p_2 \geq 0) \wedge (p_2 \leq 1)$$

szakasz, melynek az eredeti pontból widening operátorral vett kiterjesztése a

$$(p_3 = 0) \wedge (p_1 - p_2 = 0) \wedge (p_1 \geq 0)$$

félegyenes (2.5a. ábra). A kapott félegyenesen belül t_2 a

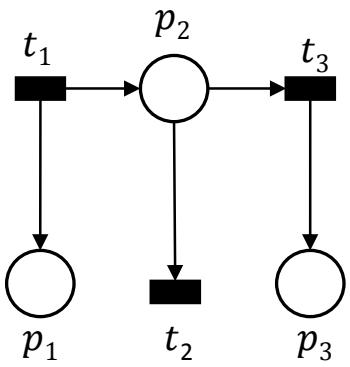
$$(p_3 = 0) \wedge (p_1 - p_2 = 0) \wedge (p_1 \geq 1)$$

félegyenesen enegedélyezett, ahonnan eltüzelve a

$$(p_3 = 0) \wedge (p_1 - p_2 = 1) \wedge (p_1 \geq 1)$$

félegyenesen lévő állapotok kerülnek be az állapottérbe. Ennek uniója a kiinduló félegyenessel

$$(p_3 = 0) \wedge (p_2 \geq 0) \wedge (p_1 - p_2 \geq 0) \wedge (p_1 - p_2 \leq 1.)$$



2.4. ábra. Egyszerű termelő-fogyasztó rendszer

A kapott állapottér-reprezentációt a kiinduló félegyenesből kiterjesztve az eredmény

$$(p_3 = 0) \wedge (p_2 \geq 0) \wedge (p_1 - p_2 \geq 0) \qquad \textit{(2.5b. ábra).}$$

Ezen a félsíkon belül t_3 engedélyezett, ahol $p_2 \geq 1$, azaz a

$$(p_3 = 0) \wedge (p_2 \geq 1) \wedge (p_1 - p_2 \geq 0)$$

félsíkon. A tüzelése (2.5c. ábra) után a kapott állapotok

$$(p_3 = 1) \wedge (p_2 \geq 0) \wedge (p_1 - p_2 \geq 1,)$$

melynek az eddigi állapottérrel vett uniója

$$(p_1 - p_2 - p_3 \geq 0) \wedge (p_2 \geq 0) \wedge (p_3 \geq 0) \wedge (p_3 \leq 1,)$$

az eredeti félsíkból kitejesztve

$$(p_3 \geq 0) \wedge (p_2 \geq 0) \wedge (p_1 - p_2 - p_3 \geq 0)$$

invariánsokkal leírható félterekkel körülhatárolt poliéder (2.5d. ábra). Az első iteráció végén ez a kapott poliéder. A kilépési feltétel nem teljesül, mivel

$$\{(0,0,0)\} \neq \{(x,y,z) | z \geq 0) \wedge (y \geq 0) \wedge (x - y - z \geq 0\}.$$

A következő iteráció során ugyanez játszódik le, azonban új állapotok már nem kerülnek be az abszrakt állapottérbe, így az eredményül kapott poliédert leíró invariánsok

$$(p_3 \geq 0) \wedge (p_2 \geq 0) \wedge (p_1 - p_2 - p_3 \geq 0.)$$

A kapott invariánsok közül a leghasznosabb (a másik kettő a Petri-háló tulajdonságaiból következik) a $p_1 \geq p_2 + p_3$, vagyis a termelő legalább annyit termel, mint amennyi a csatornában van, és amennyit a fogyasztó elfogyaszt összesen, tehát az algoritmus által kihozott eredményből látszik a rendszer jellege.

Példa 5 (Az algoritmus eredménye egy komplexebb Petri-hálóra) *A másik példa a 2.2b. ábrán látott Petri-háló, melynek állapottere*

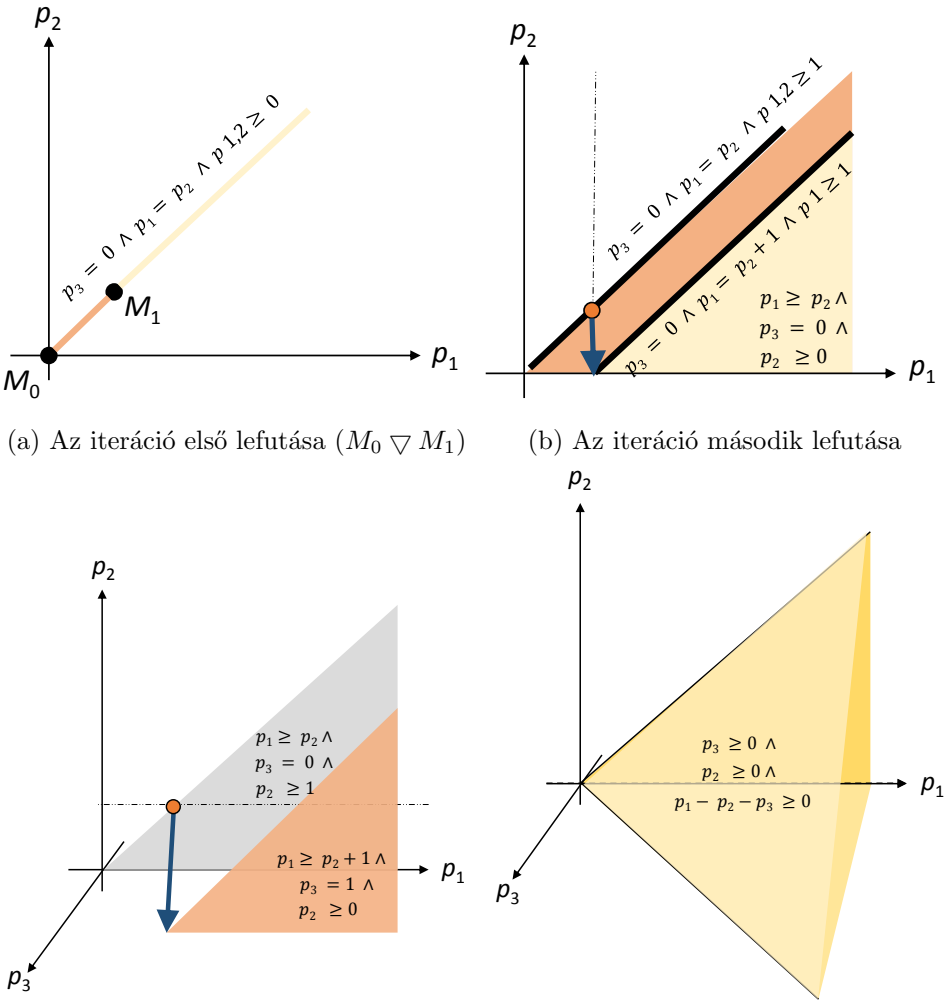
$$((P = 1) \wedge (Q = 0) \wedge (1 \leq X + Y \leq 2^Z))$$

$$\vee$$

$$((P = 0) \wedge (Q = 1) \wedge (1 \leq X + 2Y \leq 2^{Z+1}))$$

[6]. Az algoritmus lépésenkénti végigkövetését nélkülözve a kapott eredmény

$$(P + Q = 1) \wedge (Z \geq 0) \wedge (Y \geq 0) \wedge (X + Y \geq 1).$$



(a) Az iteráció első lefutása ($M_0 \bigtriangledown M_1$)

(b) Az iteráció második lefutása

(c) Harmadik iterációbeli eltolás

(d) Végeredmény

2.5. ábra. Állapottér-reprezentáció építés

Az algoritmus segítségével tehát megállapítható P és Q viszonya, valamint $1 \leq X + Y$, azonban nem derült ki, hogy ha $P = 0, Q = 1$, akkor $1 \leq X + 2Y$ is fennáll, valamint lineáris egyenlőtlenségekkel le sem írható a változók összege és kettőhatványok között fennálló viszony. Emellett bizonyítható például olyan állapotok elérhetetlensége, ahol $X = 0 \wedge Y = 0$, vagy $P + Q > 1$ illetve ha figyelembe vesszük a Petri-háló állapotaira igaz kényszereket (minden változó értéke nemnegatív egész szám), akkor $P + Q = 1$ nyomán tudjuk, hogy csak a $P = 0 \wedge Q = 1$ és $P = 1 \wedge Q = 0$ térrészek jöhetnek szóba, így logikusan szűkíthető az állapottér.

Az algoritmus áttekintése

Az az absztrakt interpretáció, valamint az algoritmus eredményesen alkalmazható Petri-hálók elérhetőségi, illetve rész-elérhetőségi problémáinak megoldásakor. Segítségével alátámasztható bizonyos állapotok, illetve állapothalmazok elérhetetlensége, valamint az alkalmazott widening operátor miatt az algoritmus időben is hatékony.

Mindemellett azonban az algoritmusnak a hozzá hasonlóakhoz képest számos hátránya van, melyek közül egyik legfőbb, hogy az alkalmazott widening operátorral kapott túlzott felsőbecslésből, valamint a poliéderek konvexitásából, illetve linearitásából adódóan nem elég precíz a kapott állapottér-reprezentáció, így sok esetben nem képes megfelelő eredményt adni. Munkám során ezen hátrányok kiküszöbölésével foglalkoztam.

3. fejezet

Hátrafelé keresés

Gráfbejáró algoritmusok között útvonalkereséshez hatékonyabbak azok, amelyek nem csak a kiindulópontból keresnek a cél felé, hanem ezzel egyidejűleg a cél irányából is hátrafelé. Ehhez azonban definiálni kell, hogy mi lehet egy adott csúcspontot (itt: állapotot) megelőző gráfcsúcs. A két irányú útkereséshez hasonló elven létrehoztam egy módszert, amely az eredeti algoritmust felhasználva a célállapotból teszteli a kezdőállapot elérhetőségét. Ebben a fejezetben ismertetem az új keresési stratégiát, illetve bemutatom, hogyan lehet ezt felhasználni az elérhetőségi probléma eldöntésének hatékonyabbá tételéhez, valamint a rész-elérhetőségi probléma megválaszolásához.

3.1. Származtatott modell előállítása

A célállapot felőli állapottér felederítés megoldásához szükséges annak a definiálása, hogy mik lehettek az egyes állapotokat megelőző állapotok. Az én módszeremben ez egy, a vizsgálandó modell alapján előállított segédmodell használatával történik. A segédmodell előállítása oly módon történik, hogy a származtatott modell strukturális viselkedése az eredetihez képest éppen „ellentétes" legyen, azaz akkor és csak akkor legyen valamely M állapotból valamely M' állapot elérhető, ha a vizsgálandó modellben M' állapotból elérhető M állapot. M-et tehát azon állapotok előzhetik meg, amelyek a származtatott modellben M-ből egy lépéssel elérhetők. A tranzíció ilyen módon működő párját *inverz tranzíciónak* (t^{-1})hívom.

A bemutatott modellező eszközökben az állapotátmenetet a tranzíciók határozzák meg, így ezek működését kell „visszafordítani". Formálisan a cél hogy $M \xrightarrow{t} M'$ állapotátmenet akkor és csak akkor legyen engedélyezett a segédmodellben, ha az eredetiben $M' \xrightarrow{t} M$ engedélyezett állapotátmenet. Lineáris tranzíciós rendszerek esetén a tranzíció hatását változónként egy $v_i \rightarrow f(v_1, v_2, \ldots, v_n)$ lineáris függvény definiálja. A segédmodell származtatásához $v'_i \rightarrow f(v'_1, v'_2, \ldots, v'_n)$ alakban keresem az

f^{-1} függvényt, viszont ez nem minden esetben megvalósítható. Olyan esetekben, mint pl. $f(v) = 0$ az azt megelőző állapot nem adható meg egyértelműen.

Fontos azonban azt is megjegyzeni, hogy itt is alkalmazható felső becslés, így elképzelhető egy olyan módszer, amely az összes elképzelhető ilyen állapotot felveszi a hátrafelé keresés során bejárt állapottérbe; azonban erre még nem alkottam konkrét algoritmust, így a hátrafelé keresés általános esetben még nem alkalmazható. Szerencsére azonban a egyszerű módon eldönthető hogy invertálató-e egy adott transzformáció, és ha igen, akkor ki is számítható az inverze.

3.1.1. Származtatott Petri-háló

Petri-hálók esetében a cél, hogy a származtatott modellben egy adott t tranzíció pontosan akkor legyen engedélyezett egy M tokeneloszlás esetén, ha létezik olyan M' tokeneloszlás, ahol az eredeti modellben t engedélyezett, és tüzelésének hatására M lesz a tokeneloszlás valamint hogy ekkor a származtatott modellben a tranzíció tüzelésének hatására M' legyen a tokeneloszlás ($M \xrightarrow{t} M' \Leftrightarrow M' \xrightarrow{t^{-1}} M$).

Inverz tranzíciós Petri-háló előállításához a következő módszert hoztam létre:

Állítás 1 (Inverz tranzíció egyszerű Petri-háló esetén) *Egyszerű Petri-hálók esetén a fenti tulajdonságokkal rendelkező, inverz tranzíciós segédmodell előállításához elég minden él helyére felvenni egy vele azonos súlyú, de ellentétes irányú élet.*

A konstrukció működését a korábban már látott példán szemléltetem. A 2.2b. ábrán is látható Petri-háló eredeti és származtatott változata a 3.1. ábrán is látható. Az eredeti Petri-háló $M_0 = (1, 0, 1, 0, 0)$ kezdőállapotból t_1 eltüzelésével $M_1 = (1, 0, 0, 1, 0)$ állapotot, M_0-ból t_2 eltüzelésével $M_2 = (0, 1, 1, 0, 0)$ állapotot kapjuk. A származtatott Petri-háló esetén M_1 állapotban t_1 engedélyezett, és tüzelésekor M_0-t kapjuk, M_2 állapotban t_2 engedélyezett, és szintén M_0-t kapjuk.

Tiltó élek esetén az inverz tranzíciós Petri-háló származtatása komplexebb feladat. Amennyiben egy helyről tiltó él vezet egy tranzícióba, a tüzelési feltétel a tokenek számát felülről is korlátozza. Mivel a tranzíció konstans számmal változtatja meg a tokenek számát, a tüzelés utáni állapotban is lehetséges felső korlátot adni a helyen lévő tokenek számára. Tiltó éleket tartalmazó Petri-hálók esetén a tiltó él nélküliekéhez képest ezt a plusz feltételt kell megragadni.

Állítás 2 (Inverz tranzíció tiltó éleket tartalmazó Petri-háló esetén) *Tiltó élekkel kiterjesztett Petri-hálók esetén a származtatott modell a következőképpen alakul:*

- *egyszerű élek helyére felveszünk egy ellentétes irányú, de azomos súlyú élet az egyszerű Petri-hálóknál látottakkal teljesen azonos módon*
- *tiltó élek súlya az egyszerű élek súlya alapján módosul:*

$$w_i(p,t)' = w_i(p,t) - w_{in}(p,t) + w_{out}(t,p).$$

A fenti egyenlet az alapján jött ki, hogy t eltüzelésének hatására $w_{out}(t,p) - w_{in}(p,t)$ számmal változik a tokenek száma, így tüzelés után ennél kevesebb token lehet az adott helyen, tehát az inverz tranzíció is csak ekkor tüzelhet.

3.1.2. Származtatott Vektor-alapú Számláló Rendszer

A Vektor-alapú Számláló Rendszerből származtatott segédmodellben csak az összeadandóvektort kell kicserélni inverz megfelelőjére. Természetesen a megoldás vektor ellentettje lesz.

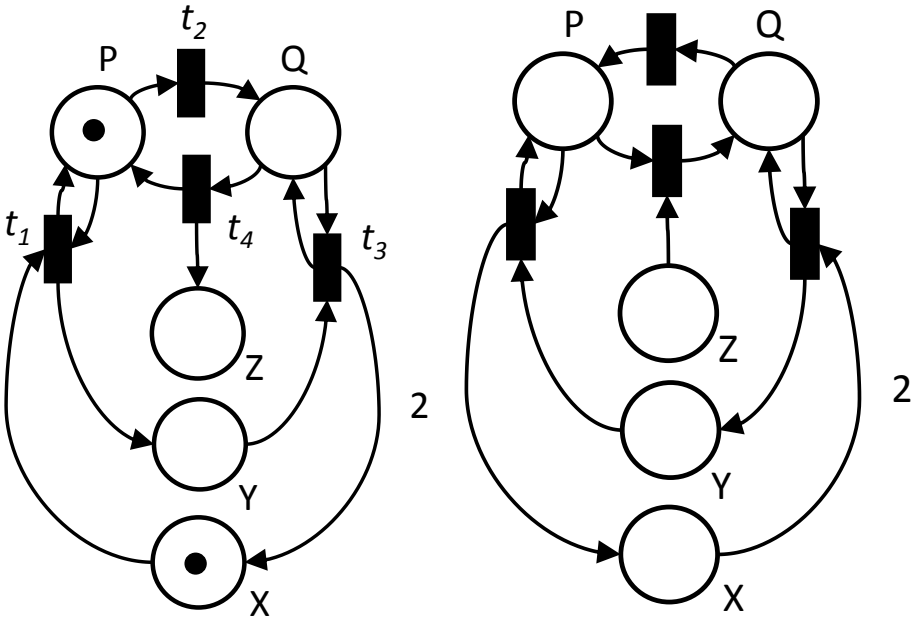
Kicsivel bonyolultabb a helyzet a vezérlési állapotokkal kiegészített Vektor-alapú Számláló Rendszerek esetében, ekkor a grafikus reprezentáció élei helyére is felveszünk egy vele ellentéte irányú élet, amelyen az összeadandó vektor ellentettje áll. Ugyanez a helyzet a lineáris tranzíciós rendszerek vezérlési állapotai esetében is. A 2.2a. ábrán látható Vektor-alapú Számláló Rendszerből származtatott segédmodell a 3.2. ábrán látható.

3.2. Az eredeti algoritmus visszafelé kereséssel kiegészítve

Az előzőekben bemutattam, hogyan lehet előállítani egy olyan származtatott modellt, amelyre igaz az, hogy az eredeti modellben pontosan akkor érhető el a kezdőállapotból a keresett célállapot, ha a származtatott modellben elérhető a célállapotból a kezdőállapot.

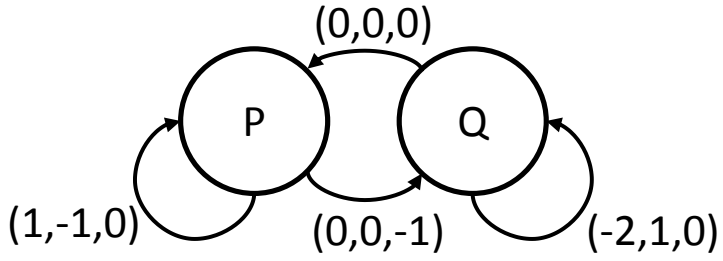
Az általam létrehozott módszer a következő: Először kiszámoljuk az absztrakt állapottér-reprezentációt, majd vesszük azt a részhalmazt, amelyre teljesülnek a feltételek. A talált kedvezőtlen állapothalmazból, mint kiinduló állapotból kiszámoljuk a poliéder-reprezentációt majd ellenőrizzük, hogy tartalmazza-e az eredeti kezdőállapotot. Ha nem, akkor tudjuk, hogy a célállapot (illetve keresett tulajdonságú állapot) nem elérhető.

Ezzel tehát olyan esetekben is meg lehet mutatni az elérhetelenséget, ahol az eredeti algoritmus nem tudta, azaz jóval precízebb megoldást adhatunk.



(a) Eredeti Petri-háló (b) Származtatott Petri-háló

3.1. ábra. Inverz tranzíciós Petri-háló előállítása



3.2. ábra. Származtatott Vektor-alapú Számláló Rendszer

Input: $PN = (P, T, E, W, M_0)$ Petri-háló, *unwanted* kedvezőtlen állapotok.
Output: $false$, vagy *unknown* attól függően, hogy sikerült-e alátámasztani az elérhetetlenséget

function BACKWARDS(PN, unwanted)
  reachable := BASIC(PN) ▷ Az eredeti algoritmus
  M_{rev} := reachable ∩ unwanted ▷ A kapott rossz állapotok
  if M_{rev}= ∅ **then return** false ▷ Ha üres, akkor nem elérhető
  end if
  PN_{rev} := inverse(PN,M_{rev}))
  from := BASIC(PN_{rev}) ▷ Hátrafelé keresés
  ▷ Ha hátrafelé nem elérhető, akkor előre sem
  if $M_0 \notin$ from **then return** false
  else return unknown
  end if
end function

3.2.1. Példa

A módszer hatékonyságát egy példán szemléltetem.

Példa 6 (Hátrafelé keresés) *A 2.2b. ábrán látható Petri-hálóról az eredeti algoritmus nem tudja bebizonyítani, hogy $M = (1, 0, 5, 0, 0)$ nem elérhető (és semelyik más, lineáris invariáns keresésére alkalmas algoritmus sem, mivel a valódi állapottér konvex burkának része M), tehát a fenti algoritmus futtatása során $M_{rev} = M$ lesz.*

Visszafelé, a 3.1b. ábrán látható Petri-háló struktúrából M kezdőállapottal fog keresni az algoritmus, viszont itt egyik inverz tranzíció sem engedélyezett, tehát a segédmodellben $R(M) = M$. Az inverz tranzíciós Petri-hálóban M-ből M_0 nem elérhető, tehát az eredetiben M_0-ból M sem elérhető.

Ezzel a módszerrel tehát kiküszöbölhetőek az eredeti algoritmus olyan hátrányai, amelyek a poliéderek konvexitásából, vagy az algoritmus által alkalmazott widening operátor használatából következnek. A fenti leírásból az is látszik, hogy akár más absztrakciós módszerekre is alkalmazható. Szintén újítás, hogy míg az eredeti és a hozzá hasonló algoritmusok csak akkor tudnak biztonságosságot bizonyítani, ha az általuk talált állapottér-reprezentáció nem tartalmazza a kedvezőtlen állapotot, ezzel a módszerrel a keresett állapot figyelembevételével történik a bizonyítás, így akár finomítani is lehet vele az absztrakciót.

4. fejezet

Absztrakció-finomítás

Az előző fejezetekben bemutattam a felhasznált algoritmus hátrányait, hangsúlyt fektetve a túlzott felülbecslés elkerülésére. Munkám során több megközelítést is találtam az absztrakció finomítására. Ebben a fejezetben ismertetem az algoritmus azon módosításait, amelyekkel precízebb eredmény érhető el; bemutatok egy módszert, amely segítségével az algoritmust vezérlési állapotokat tartalmazó modellekre is lehet alkalmazni; valamint leírok egy megközelítést, amely az algoritmust használja fel az explicit állapottérbejárással való elérhetőségi probléma eldöntés hatékonyabbá tételére.

4.1. Fókuszálás

Az algoritmusban a „legdurvább" felülbecslés (legtöbb, az állapottér-reprezentációban szereplő, valójában nem elérhető állapot bevezetése) a widening operátor alkalmazása hatására következik be. Akár a második elérhető állapot felfedezése után végtelenné terjesztheti ki az állapottér-reprezentációt. Számos módszer született widening operátorok által hamisan behozott állapotok kiszűrésére, illetve a precízió javítására, melyek közül az egyik leghatékonyabb a *fókuszálás* (narrowing operátor), mely a widening operátor ellentéte abból a szemponból, hogy míg a kiterjesztés növeli, a fókuszálás csökkenti az állapottér-reprezentáció méretét, azoban sajnos konvex poliéderekhez még nem létezik narrowing operátor.

Más, poliéderek esetén is alkalmazható megoldások közé tartoznak a widening operátor alkalmazásának késleltetése, valamint egy időben kevésbé hatékony, viszont precízebb widening operátor alkalmazása, azonban a gyakorlatban az a tapasztalat, hogy ezekkel a módszerekkel is csak késleltetni lehet a korlátlan mennyiségű nem elérhető állapot állapottérbe kerülését.

Egyik megközelítésem, amely az egyszerre megjelenő végtelen felesleges állapotot hivatott elkerülni, az algoritmus azon hiányosságát igyekszik pótolni, hogy a widening operátor alkalmazása után semmilyen ellenőrzést

nem végez, hogy egyáltalán indokoltan kerülnek-e be az új állapotok az állapottérbe. Az alapgondolat a következő:

Ahogy az pl. a 2.5a. ábrán is látható a második állapot (pont) megjelenésével a widening operátor az állapottér-reprezentációt (szakasz) végtelenné terjeszti ki (félegyenes), és onnantól kezdve minden állapotot elérhetőnek tekint. A logika emögött az, hogy ha egyszer el lehetett tüzelni a tranzíciót, akkor valószínűleg többször is, így a többi állapot is megjelenhet. Amennyiben azonban a tranzíció a kapott állapotok nem mindegyikén engedélyezett, hanem csak véges esetben (tiltó él miatt például) akkor a többi állapot mindenképpen hamisan került be az állapottérbe. Amennyiben a widening operátor segítségével kiterjesztett állapottéren még ellenőrizzük, hogy hol engedélyezett az adott tranzíció, majd ezekről eltüzeljük, és az állapottérhez csak az így kapott új állapotokat adjuk hozzá, akkor elkerülhető az indokolatlan állapotok bekerülése az állapottérbe.

Bár a módszer első sorban a tiltó éleket tartalmazó Petri-hálók állapotterének precízebb becslését segíti elő, egyszerű Petri-hálók esetében is eredményes; például kölcsönös kizárást biztosító algoritmusokat leíró Petri-hálókra az eredeti algoritmus éppen a fentiek miatt nem tudott a kívánt eredménnyel szolgálni, azonban ezen módosítás alkalmazásával az algoritmus eredményesen bizonyította a kölcsönös kizárás fennállását. Ezzel a módosítással az algoritmus rendkívül precíz eredményt ad, azonban a widening operátor nem megfelelő alkalmazása miatt a terminálódás ebben az esetben már nem garantált.

Az algoritmus terminálódása mindenképpen szükséges, ezért ugyanezen ötlet felhasználásával létrehoztam egy olyan megoldást is, ahol ez garantált, viszont ez sokkal kisebb mértékben finomítja az absztrakciót. Ehhez nem az algoritmus futása közben, hanem a fixpont elérése után végzem el a fenti műveletet minden egyes tranzícióra. Ekkor azonban a pontosabb válasz érdekében akár a művelet közben kölön poliédereket lehet definiálni az egyes tranzícióknak, ezzel kiküszöbölve bizonyos, a poliéder konvexitásából adódó elérhetetlen állapotokat.

Ez a módszer is finomítja az absztrakciót, azonban az első, terminálódást nem garantáló megközelítésem sokkal eredményesebb volt ezen a területen (könnyen belátható, hogy a második módszerrel kapott poliéder minden esetben tartalmazza az első módszerrel kapott poliédert). Az optimális felhasználás érdekében kidolgoztam egy olyan módszert, amely a fenti két megközelítés együttes alkalmazásával lehetővé teszi nagymértékű precízió elérését az algoritmus terminálódásának biztosítása mellett. Ehhez a Petri-háló, és a kérdés mellett azt is meg kell adni az algoritmusnak bementi paraméterként, hogy az első módszer maximálisan hányszor alkalmazható, mielőtt a widening operátor „érvényre juthat”, és terminálódhat az algoritmus (a fix pont megtalálása után alkalmazható az utóbbi módszer). Az algoritmus pszeudo kódja a következő:

Input: $PN = (P, T, E, W, M_0)$ Petri-háló, max felső korlát
Output: *polyhedra* konvex poliéderek halmaza, melyek uniója PN állapotterének felülbecslése

```
function REFINED(PN,max)
    reachable := {M_0}
    iteration := 0
    repeat
        old := reachable
        for all transition t ∈ T do
            enabling:=enablingCondition( t)
            enabled := reachable ∩ enabling
            if enabled =∅ then continue
            end if
            next := fire(t, enabled)
            temp := reachable ▽ ( reachable ∪ next )
            if iteration ≤ max then
                enabled := temp ∩ enabling
                next := fire(t, enabled, F)
                reachable := reachable ∪ enabled ∪ next
                iteration := iteration + 1
            else
                reachable := temp
            end if
        end for
    until  reachable = old ;
    polyhedra := {M_0}
    for all transition t ∈ T do
        enabling := enablingCondition( t)
        enabled := reachable ∩ enabling
        next := fire(t, enabled)
        polyhedra += enabled
        polyhedra += next
    end for
     return polyhedra
end function
```

4.2. Fókuszálás a vezérlési állapotok szerint

Ebben a részben bemutatom, hogy hogyan lehet az algoritmust kiterjeszteni Vektor-alapú Számláló Rendszerekre, illetve Lineáris tranzíciós rendszerekre, majd megmutatom, hogyan lehet a vezérlési állapotokat precíziónövelésre felhasználni.

Az algoritmus más adatszerkezetekre való kiterjesztése során a cél az információveszteség minimalizálása, azaz hogy az adott modellezési eszköz sajátosságai figyelembevételével, rájuk építve végezzük minél precízen a modellellenőrzést. Ebben a részben ezt mutatom be.

4.2.1. Az algoritmus kiterjesztése Vektor-alapú Számláló Rendszerekre

Az algoritmus egyszerű Vektor-alapú Számláló Rendszerre minden változtatás nélkül felhasználható, a vezérlési állapotokkal kiterjesztett esetben azonban a minél precízebb eredmény elérése érdekében szükség van azok kezelésére. Megoldást jelenthet például az aktuális vezérlési állapot eltárolása egy erre dedikált változóban, vagy a Vektor-alapú Számláló Rendszer leképzése Petri-hálóvá; ezekben az esetekben azonban gyakran éppen a lényeges információ absztrahálódik el az algoritmus során. A következő módszerrel azonban az algoritmus kiterjeszthető több vezérlési állapotot tartalmazó modellekre az egyes vezérlési állapotokra jellemző invariánsok megtartásával.

A fő gondoloat az, hogy az algoritmus során nem egy poliédert bővítünk, hanem minden vezérlési állapothoz külön-külön felépítjük az állapottér-reprezentációt, ezáltal mindegyiknek saját invariánsokat előállítva, hogy az ellenőrzés során akár vezérlési állapot-specifikus kérdésekre is választ adhassunk. Megjegyzendő, hogy ezt az ötletet a szoftvertesztelés témaköréből merítettem. Ott különböző programpontokon (a vezérlési állapotoknak megfelelő *location*) ellenőrzik, hogy milyen értéket vehet fel a változó, azzal a különbséggel, hogy ott tranzíciótüzelések helyett a programkód szimbolikus végrehajtásával következtetnek az új állapotokra.

Ezt a módszert alkalmazva bizonyos, a poliéder konvexitásából adódó hátrányok is kiküszöbölhetők, valamint ha a (rész-)elérhetőségi kérdésre adott válasz *igen*, akkor megadhatóak azok a vezérlési állapotok, ahol a kérdéses kedvezőtlen állapotok feltételezetten előállhatnak. Ehhez azonban néhány egyéb módosítást is eszközölni kell az eredeti algoritmuson.

- Kezdetben a kiinduló vezérlési állapothoz tartozó poliéder (amely a kezdőállapotot tartalmazza) kivételével mindegyik üres. Értelemszerűen később ezekbe is kerülhetnek állapotok, de akár az is előfordulhat, hogy az algoritmus segítségével bizonyítani lehet, hogy a rendszer bizonyos vezérlési állapotokba soha nem juthat el.

- A kapott új állapotok nem feltétlenül ugyanabba a poliéderbe kerülnek be, ahonnan az engedélyezett állapotokat kiválasztottuk, hanem abba, amelyik a Vektor-alapú Számláló Rendszer gráf-reprezentációjában a tranzíciónak megfelelő él végpontjában található vezérlési állapothoz tartozik.

- Az előző változtatás miatt a widening operátor használata logikailag értelmét veszti, így a percíz becslés érdekében a kiterjesztés körülményein is érdemes módosítani - azon tranzíciók esetében, amelyek eltüzelésének hatására vezérlési állapotváltás történik, késleltetni kell a widening operátor alkalmazását néhány lépéssel.
- Az elérhetőségi problémát minden poliéderre külön-külön ellenőrizni kell, és csak akkor lehet *nem* választ adni, ha ez minden vezérlési állapotra együttesen teljesül.

Input: $VASS$ a vezérlési állapotokkal kiterjesztett Vektor-alapú Számláló Rendszer, S a vezérlési állapotainak halmaza
Output: *reachable* az absztrakt állapottér vezérlési állapotok szerint poliéderekre bontva

```
function basicForVASS(VASS)
    reachable = ∅                          ▷ A poliédereket tartalamazó halmaz
    for all s ∈ S do
        polyhedron = ∅
        s ← polyhedron
        reachable += s.polyhedron
    end for
    init.polyhedron += M_0                                  ▷ kezdőállapot
    repeat
        old := reachable
        for all transition t ∈ T do
            enabling := enablingCondition( t)
            s := fromState(t)
            enabled := s.polyhedron ∩ enabling
            if enabled =∅ then continue
            end if
            next := fire(t, enabled)
            s := toState(t)
            s.polyhedron := s.polyhedron ▽ ( s.polyhedron ∪ next )
        end for
    until  reachable = old
     return reachable
end function
```

4.2.2. Az algoritmus kiterjesztése Lineáris Tranzíciós Rendszerekre

Az eredeti algoritmusban nem volt konkrétan leírva, hogy az engedélyezett állapotok kiválasztása, illetve a tranzíció tüzelése után kapott állapotok

kiszámolása hogyan történjen; ezek konvex poliéderekre vett értelmezése a Petri-háló alapú viselkedésmodellek működési jellegéből adódott. Ennek következtében lineáris tranzíciós rendszerekre újra definiálni kell. (A vezérlési állapotok kezelése az előző algoritmushoz képest változatlan.)

Az őrfeltételek már nem csupán egy-egy változó intervallumára vonatkozhatnak, hanem tetszőleges változóra felírt lineáris kényszerek is lehetnek, azonban a Petri-hálóknál látott engedélyezettségi poliéder ebben az esetben is felírható, így az engedélyezett állapotok kiválasztásának elve változatlan. A tranzíciók hatása ezzel szemben már nem feltétlenül írható le a Petri-háló alapú modelleknél látott térbeli eltolással, éppen azért mert a változó új értéke nem csak a saját régi értékétől függhet. A megoldást a lineáris transzformáció jelenti, mely éppen arra alkalmas, hogy a transzformált koordinátákat az eredeti koordináták lineáris függvényeként állítson elő. Petri-háló alapú modellek esetében egy vektorral írtuk le a tranzíció hatását; a lineáris transzformáció egy mátrix segítségével adható meg. A tranzíció invertálhatóságának eldöntése és az inverz tranzíció előállíatása ez alapján történik.

4.2.3. Az ötlet felhasználása absztrakciófinomításhoz

Könnyen belátható, hogy a módosított algoritmus a vezérlési pontok figyelembevételével jóval precízebb eredményt ad, mintha az eredeti algoritmus (vagy akár valamely, a dolgozatban leírt módosított változat) végrehajtásával a vezérlési pontokat nélkülözve, csupán a változók figyelembevételével próbálnánk becsülni az állapotteret. Ezen a gondolatmeneten elindulva nyilvánvalóvá válik, hogy saját vezérlési állapotok definiálásával (akár Petri-hálókhoz is) finomítható az absztrakció.

Ehhez először meg kell adni a vezérlési állapotokat, majd minden tranzícióra fel kell írni hogy melyik vezérlési állapotból melyik vezérlési állapotba vezet. Könnyen előfordulhat, hogy egy adott tranzíció több vezérlési állapotban is engedélyezve van, illetve hogy egy adott vezérlési állapoton belül különböző engedélyezett állapotokból eltüzelve különbőző vezérlési állapotokba kerül át. Ezek a problémák egyszerűen áthidalhatók a tranzíció többszörözésével, illetve az engedélyezési feltételek módosításával a „klón"-tranzíciókon, így a tüzelés minden részesetének megfeleltethetünk egy-egy tranzíciót, amelyhez így már egyértelműen hozzárendelhető a kiinduló, illetve az elért vezérlési állapot. Ha meglévők mellé szeretnék új vezérlési állapotokat definiálni, askkor azok szétbontásával, és a tranzíciók fenti módon leírt hozzáigazításával tehető meg.

Következő kérdés, hogy mi alapján válasszunk szét vezérlési állapotokat. A hátrafelé keresésről szóló fejezetben kifejtettem, hogy miért hasznos a keresési feltétel felhasználása az absztrakció alkalmazásakor a precízebb eredmény érdekében. Ezen az elven elindulva az új vezérlési állapotokat

érdemes a kérdés alapján bevezetni. Egyszerű kérdés esetén ez akár automatizálva is történhet.

4.3. Egyéb módszerek invariánskeresésre

Invariánsok keresésére számos módszer létezik, köztük enumerációs, transzformációs és strukturális technikák. Az absztrakt interpretáció csak egy ezen absztrakciós technikák közül. Annak ellenére, hogy ezeket általában „versenyeztetni" szokták egymással (különböző példák segítségével tesztelik, hogy melyik ad precízebb megoldást), valójában az absztrakciós módszerek által alkalmazott felsőbecslésből adódóan a legprecízebb megoldást az összes hasonló technika együttes alkalmazása adja. Dolgozatomnak nem célja minden ilyen módszert ismeretetni, azonban egy példán bemutatom, hogy hogyan érdemes felhasználni a már ismert invariánsokat az algoritmus során.

4.3.1. Induktív invariánsok generálása

Sankaranarayanan, Sipma és Manna bemutatták [11], hogyan állítható elő az összes induktív invariáns egy egyszerű (tiltó él nélküli) Petri-hálóra egy egyszerű és hatékony algoritmus segítségével.

Definíció 7 (Induktív kényszer) *Egy* Φ *kényszer akkor és csak akkor induktív egy tranzíciós rendszerben, ha (1. Inicializáció) teljesül a kezdőállapotban és (2. Folyamatosság) ha egy tranzíció tüzelése előtt teljesül, akkor a tüzelés után is fog.*

Könnyű bebizonyítani, hogy minden induktív kényszer egyben invariáns kényszer is az adott rendszerre. Az induktív invariáns tehát egy olyan lineáris invariáns, amely igaz a kezdőállapotban, és amely teljesülését egyetlen tranzíció tüzelése sem változtatja meg. Induktív invariánsok nagyon hatékonyak az ellenőrzés során, mivel az induktivitás ellenőrzéséhez nincs szükség az állapottér bejárására.

Mivel a Petri-háló tiltó éllel való kiegészítése „csökkenti" az állapotteret, ezért az algoritmus a tiltó éllel ellátott Petri-hálókra is érvényes induktív invariánsokat állít elő, azonban ebben az esetben már nem teljes. Ugyanez vonatkozik a vezérlési állapotokkal való kiterjesztésre, illetve általános esetben is léteznek módszerek – a Petri-hálókhoz generált induktív invariánsok előállítása a lineáris tranzíciós rendszerekhez használható módszer [3] továbbgondolása.

4.3.2. Ismert invariánsok felhasználása az algoritmus hatékonyságának növeléséhez

Amennyiben valahonnan (például a fent említett invariáns generáló technika segítségével) tudjuk, hogy bizonyos invariánsok mindenképpen tel-

jesülnek, akkor ezekkel tudjuk finomítani az algoritmus által talált absztrakciót. Kézenfekvő megoldás a fix pont megtalálása *után* hozzáadni a talált invariánsokhoz az eddig ismerteket (tehát venni a két poliéder metszetét), azonban hatékonyabb megoldás érhető el ha már az algoritmus futtatása közben figyelembe vesszük, hogy a megadott egyenlőtlenségeknek mindenképpen teljesülnie kell.

Erre a precízió szempontjából optimális megoldás az algoritmus minden lépése után elmetszeni a poliédert az ismert invariánsokkal, ez azonban nem hatékony. A kettő közötti kompromisszumos megoldás a legtöbb nem elérhető állapotot behozó lépés, a kiterjesztés után finomítani az absztrakciót.

A modosított algoritmus a pszeudokódja a következő:

Input: $PN = (P, T, E, W, M_0)$ Petri-háló, *known* poliéder, amely az ismert invariánsokat írja le

Output: $R(PN, M_0)$-ra adott felső becslés (itt: a poliédert leíró invariánsok)

```
function basicWithKnown(PN,known)
    reachable = {M_0}
    repeat
        old := reachable
        for all transition t ∈ T do
            enabling:=enablingCondition( t)
            enabled := reachable ∩ enabling
            if enabled =∅ then continue
            end if
            next := fire(t, enabled)
            reachable := reachable ▽ ( reachable ∪ next )
            reachable := reachable ∩ known                ▷ Szűkítés
        end for
    until reachable = old ; return reachable
end function
```

4.4. Ellenpélda előállítás útvonalkereséssel

Az absztrakció alapú módszerek időben hatékonyak, azonban a felülbecslés miatt nem tudnak elérhetőséget bizonyítani. Ezzel szemben az explicit állapottérbejáró algoritmusok képesek pontosan megadni az elérhető állapotok halmazát a lehetséges elérési útvonalakkal együtt, végtelen állapottér esetén azonban soha nem terminálódnak. A két mószer együttes alkalmazásával azonban lehetséges olyan algoritmust létrehozni, amely az

absztrakciót alkalmazva növeli az explicit állapottérbejáró algoritmus időbeli hatékonyságát a kérdés szempontjából lényegtelen utak elvágásával.

4.4.1. Algoritmus

Az explicit állapottérbejáró algoritmus valójában nem más, mint az elérhetőségi gráf feltérképezése a háttérismereteknél leírt módszerrel. A gyakorlatban a végtelen állapottér miatt szélességi, vagy iteratív mélyülő bejárást alkalmaznak, hogy szabályozni tudják a bejárás mélységét. A módszer lényege a következő.

A kezdőállapotból kiindulva, minden talált állapotban ellenőrizzük, mely tranzíciók vannak engedélyezve, majd ezekhez behúzunk egy-egy gráfélet, melynek végére a tranzíció tüzelése után kapott állapotot jelölő csúcs kerül. A talált állapotokra ugyanezt elvégezzük. Ha találunk egy megfelelő célállapotot, megállunk, és visszatérünk az elérési úttal. Ha olyan állapotot találunk, amelyet a szélességi keresés korábbi szintjén már ellenőriztünk, akkor azt figyelmen kívül hagyhatjuk, de akár be is húzhatunk egy élet. Ha már nincs új állapot, nemleges válasszal térünk vissza, ez azonban végtelen állapottér esetén nem fog megtörténni.

Éppen ezért szokták korlátozni valamilyen módon a gráfbejárás (keresés) során a keresőfa mélységét. Ha a futás során az algoritmus elér ebbe a mélységbe *bizonytalan* (unknown) válasszal tér vissza. A keresőalgoritmus tehát az elérési út (*trace*) megadásával alátámasztott biztos igen, vagy bizonytalan, míg az absztrakt interpretációs algoritmus invariánsokkal alátámasztott biztos nem, vagy bizonytalan válasszal térhet vissza. A kettő együttes alkalmazásával elérhető, hogy biztos igen és biztos nem választ is tudjon adni az algoritmus. A módszer a következő:

A kezdőállapotból kiindulva először az absztrakt interpretációs algoritmust alkalmazzuk. Csak akkor lépünk tovább, ha nem sikerült elérhetetlenséget bizonyítani. A tüzelések után (amennyiben nem értünk el kívánt célállapotba) a kapott állapotokat kezdőállapotként felhasználva futtatjuk az absztrakt interpretációs algoritmust. Az iteráció során csak abba az irányba mélyítjük az elérhetőségi gráfot, ahol nem sikerült elérhetetlenséget bizonyítani. Ha ilyen állapot nincs, biztos nem válasszal térünk vissza. Ha elértünk a feltételeknek megfelelő célállapotba, biztos igen válasszal térünk vissza. A megadott maximális mélységet elérve a válasz bizonytalan, viszont ekkor is meg lehet adni, hogy mely elérési út folytatásánál elképzelhető az állapot elérése. A pszeudokód a következő:

Input: $PN = (P, T, E, W, M_0)$ Petri-háló, *unwanted*, a kedvezőtlen állapotok halmaza, *max* a keresés maximális mélysége

Output: *true* ha az elérhetőséget, $false$ ha az elérhetetlenséget sikerült alátámasztani, *unknown* egyébként

```
function COMPLEX(PN, unwanted, max)
    root := M_0                          ▷ A kezdőállapot lesz a gráf gyökere
    unsafeLeaves := {root}               ▷ A még nem ellenőrzött állapotok
    graph := {root}
    iteration := 0
    while iteration ≤ max do
        for all leaf ∈ unsafeLeaves do
                                ▷ Ha a csúcs kedvezőtlen állapot, visszatérünk
            if leaf ∈ unwanted then return trace(graph, leaf)
            end if
                                          ▷ Meghívjuk az eredeti algoritmust
            reachable := BASIC((P,T,E,W, leaf))
            M_rev := reachable ∩ unwanted
                                ▷ Ha elérhetetlen, nem kell tovább ellenőrizni
                        ▷ Egyébként egy szinttel mélyebbre kell menni a fában
            if M_rev= ∅ then
                unsafeLeaves -= leaf
            else
                unsafeLeaves -= leaf
                for all t ∈ enabledTrasitions(leaf) do
                    next := fire(t,leaf)
                    leaf += child(t, next)
                    unsafeLeaves += next
                end for
            end if
        end for
                            ▷ Üres a halmaz esetén az elérhetetlenség bizonyított
        if unsafeLeaves = ∅ then return false
        end if
        iteration = iteration +1
    end while
     return unknown
end function
```

Ennek az algoritmusnak is léteznek különböző módosításai a hatékonyság érdekében. Többek között:

- a keresés mélysége helyett lehet korlátozni az explicit bejárt állapotok számát (ebben az esetben ez talán indokoltabb), a futás idejét, stb.
- az absztrakt interpretációs algoritmust nem érdemes minden szinten, csak megadott számú szint explicit kifejtése után futtatni.
- a körök elkerülése érdekében lehet ellenőrizni, hogy a tüzelés során kapott állapottal nem találkoztunk-e már, és ha igen, akkor azt figyelmen kívül lehet hagyni

- eseleges további analízishez lehet talált hibás állapot után is továbbfuttatni az algoritmust, így adott esetben kiderülhet, hogy csak néhány elérhető kedvezőtlen állapot van, ami akár kivételként is kezelhető.

4.4.2. Értékelés

Az eddigi algoritmusokhoz képest újítás, hogy már biztos igen válasz is adható, valamint hogy ez az algoritmus bizonytalan válasz esetén is hasznos információkkal tud szolgálni az elérési utat illetően.

Példa 7 (Kombinált algoritmus alkalmazása) *Vegyük a 2.2b. ábrán látható Petri-hálót, és a 4.1. ábrán figyeljük meg az algoritmus futását az* $M_0 = (1, 0, 1, 0, 0)$ *kezdőállapottal, és* $M = (0, 1, 2, 0, 0)$ *célállapottal. Az állapotgráf csúcsait az állapotok megjelenési sorrendje szerint számoztam, föléírva az egyre finomodó állapottér-reprezentációt. Az unknown eredményt kérdőjel, az elérhetetlenség alátámasztását az adott állapotban épülő állapottér-reprezentációban pipa, a kedvezőtlen állapotokat leíró feltételek teljesülését az adott állapotban x jelzi. Az algoritmus futása ezzel a jelölésrendszerrel a következő.*

Kezdetben (I. iteráció) az elérési gráf csak $M_0 \neq M$*-et tartalmazza, innen az eredeti algoritmus eredménye a már korábban látott*

$$(P + Q = 1) \wedge (Z \geq 0) \wedge (Y \geq 0) \wedge (X + Y \geq 1.)$$

Ez nem zárja ki a M*-et, így az engedélyezett tranzíciók* (t_1, t_2) *tüzelése során kapott* $M_1 = (1, 0, 0, 1, 0)$ *illetve* $M_2 = (0, 1, 1, 0, 0)$*,* $M_{1,2} \neq M$ *állapotokban vizsgálunk elérhetőséget. A II. iterációban* M_1 *állapotból futtatva az algoritmust megjelenik egy erősebb invariáns:*

$$X + 2Y \geq 2.$$

M_2 *állapotból az eredmény* M_0*-hoz képest változatlan. Egyik esetben sem bizonyított az elérhetetlenség, így a keresés továbbmélyül.* M_1 *állapotban csak* t_2 *tranzíció engedélyezett, tüzelése után* $M_3 = (0, 1, 0, 1, 0) \neq M$ *lesz a tokeneloszlás.* M_2 *állapotban csak* t_4 *engedélyezett, tüzelése után* $M_4 = (1, 0, 1, 0, 1) \neq M$ *lesz a tokeneloszlás. A III. iterációban az algoritmust* M_3*-ból futattva az eddigiek mellett*

$$Z - P \geq 0$$

invariáns is megjelenik, ami nem zárja ki M*-et; azonban* M_4 *kezdőállapotból*

$$Z \geq 1$$

is teljesül, ami viszont nem igaz M-re, így innen már nem kell továbbfinomítani az állapotteret. Ez azonban nem jelenti azt, hogy az állapot nem elérhető, M_3-at még ellenőrizni kell.

M_3 állapotban t_3 engedélyezett, azt eltüzelve pedig éppen $M_5 = (0, 1, 2, 0, 0) = M$ állapotot kapjuk, ezért a IV. iterációban egyből visszatérünk a $t_1 \to t_2 \to t_3$ tüzelési sorozattal.

Megjegyzendő, hogy M_2 kezdőállapotból indítva a kombinált algoritmus először nem tudná biztosan állítani az elérhetőséget, majd t_4 nyomán M_4-ből már sikerülne alátámasztani az állítást.

A kombinált algoritmus előnyei tehát a következők:

- Képes *igen* választ adni, amit a keresett állapothoz vezető elérési úttal támaszt alá
- Képes elérhetetlenséget bizonyítani, ahol az eredeti algoritmus nem tudott.

A módszer más, hasonló absztrakciós módszerek esetén is alkalmazható.

4.5. Komplex iterációs stratégia

A tervezés során fontos szempont, hogy mely algoritmust, illetve algoritmusokat érdemes implementálni optimális hatékonyság, illetve maximális precízió elérése érdekében, illetve hogy ezek mennyire kompatibilisek egymással. Ezeket szem előtt tartva megalkottam egy olyan megoldást, amely az összes korábban bemutatott módszert egyesíti egyetlen algoritmusban. Ennek működése a következő:

- Nulladik lépésként a modellt a kérdés alapján definiált új vezérlési állapotokkal kiegészítjük, módosítjuk. Ha lehetséges előállítjuk a származtatott segédmodellt is.
- Kiindulunk a kezdőállapotból.
- A lineáris tranzíciós rendszerekre kiterjesztett eredeti algoritmust a fókuszálással, az ismert invariánsok felhasználásával, illetve egyéb precízionövelő megoldásokkal alkalmazva poliédert építünk.
- Ha az elérhetetlenség teljesül, visszatérünk nem válasszal, és az ezt bizonyító invariánsokkal; ellenkező esetben folytatódik a program futása.
- Ha invertálható a modell, akkor felépítjük a az előbb talált, a kényszereknek megfelelő állapothalmazzal, mint kezdőállapothalmazzal ellátott segédmodell állapottér-reprezentációját. Amennyiben ez nem tartalmazza a kezdőállapotot, visszatérünk nem válasszal. Különben folytatódik az algoritmus.

- A kezdőállapotból minden lehetséges tranzíciót eltüzelünk. Ha valamelyik kapott állapot megfelel a feltételnek visszatérünk igen válasszal, és az állapothoz vezető elérési úttal. Különben megjegyezzük az elérhetőségi gráfot.
- Minden kapott állapotra lefuttatjuk az algoritmust esetlegesen a hátrafelé kereséssel, majd ha nem kaptunk kedvező megoldást tovább iterálunk.
- A program futása háromféle módon érhet véget.
  - Ha egy ponton minden elérési úton nem a válasz, akkor visszatérünk nem válasszal.
  - Ha olyan állapotot találunk, ami megfelel a keresési feltételeknek, visszatérünk igen válasszal.
  - Ha elértük az iteráció maximális mélységét, vagy egyéb korlátozást, visszatérünk bizonytalan válasszal, illetve az eddigi eredményeinkkel.

A kombinált algoritmus pszeudokódja a következő:

Input: $LTS = (L, T, l_0, \Theta)$ Lineáris tranzíciós rendszer, *unwanted*, a kedvezőtlen állapotok halmaza, max a keresés maximális mélysége, $maxref$ fókuszálás maximláis száma, *known* az ismert invariánsok

Output: *true* ha az elérhetőséget, $false$ ha az elérhetetlenséget sikerült alátámasztani, *unknown* egyébként

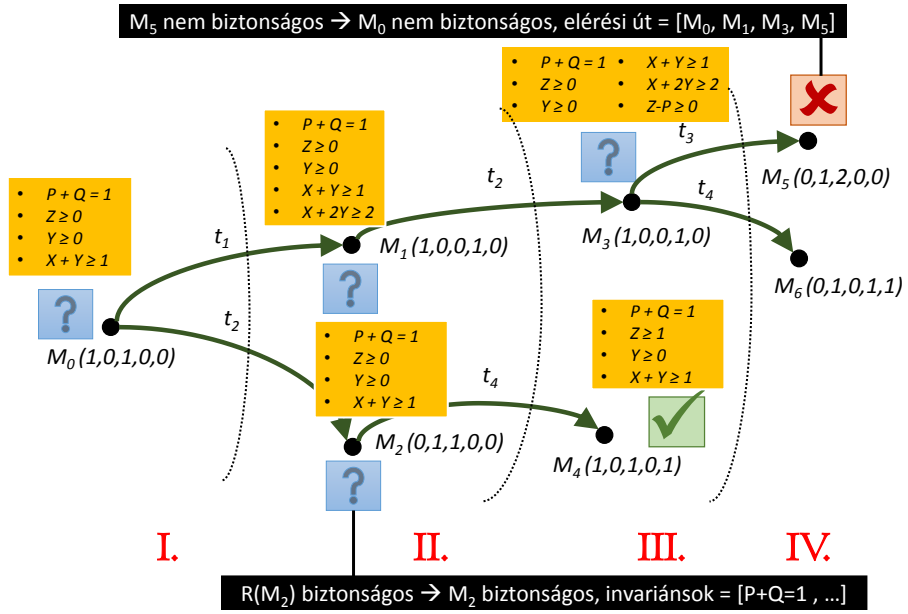
```
function COMBINED(LTS, unwanted, max, maxref, known)
    if invertable(LTS) then
        LTSrev = inverse
    end if
    root := Θ
    unsafeLeaves := {root}
    graph := {root}
    iteration := 0
    while iteration ≤ max do
        for all leaf ∈ unsafeLeaves do
            if leaf ∈ unwanted then return trace(graph, leaf)
            end if
                                              ▷ Elenőrzés kétirányú kereséssel
            result := false
            if invertable(LTS) then
                result := BACKWARDSFORLTS(LTS, LTSrev, leaf, unwanted, maxref, known)
            end if
```

```
            if result = false then
                unsafeLeaves -= leaf
            else
                unsafeLeaves -= leaf
                for all t ∈ enabledTrasitions(leaf) do
                    next := fire(t,leaf)
                    leaf += child(t, next)
                    unsafeLeaves += next
                end for
            end if
        end for
        if unsafeLeaves = ∅ then return false
        end if
        iteration = iteration +1
    end while
     return unknown
end function
```

Input: $LTS = (L, T, l_0, \Theta)$ Lineáris tranzíciós rendszer, $LTSrev$ LTS inverze, M a kiinduló állapot, *unwanted*, a kedvezőtlen állapotok halmaza, $maxref$ fókuszálás maximláis száma, *known* az ismert invariánsok

Output: $false$, vagy *unknown* attól függően, hogy sikerült-e alátámasztani az elérhetetlenséget

```
function BACKWARDSFORLTS(LTS, M, unwanted, maxref, known)
    reachable := BASICFORLTSWITHREFINEMENTS(LTS, M, maxref, known)
    M_rev := reachable ∩ unwanted                ▷ A kapott rossz állapotok
    if M_rev= ∅ then return false                ▷ Ha üres, akkor nem elérhető
    end if
    from := BASICFORLTSWITHREFINEMENTS(LTSrev,M_rev, maxref, known)
                                 ▷ Ha hátrafelé nem elérhető, akkor előre sem
    if M_0 ∉ from then return false
    else return unknown
    end if
end function
```



4.1. ábra. Kombinált algoritmus példa vizuális megjelenítése

5. fejezet

Implementáció

Ebben a részben bemutatom az implementációhoz felhasznált környezeteket, valamint az architektúra elemeit, majd ismeretetem annak az algoritmusnak a pszeudokódját, amely az összes kitalált módosítást együttesen alkalmazza.

5.1. Keretrendszer

Az implementáció alapjául szolgáló eszközök a következőek:

- A TTMC (Timed Transition systems Model Checker) egy tanszéken fejlesztett eszköz, amely időzített tranzíciós rendszerek modellellenőrzését teszi lehetővé. Többek között definiál egy nyelvet, amely segítségével a tranzíciós rendszerek könnyen leírhatók. Az algoritmusokat ezen a nyelven leírt lineáris tranzíciós rendszerek ellenőrzésére implementáltam a TTMC egy pluginjaként.

- Az algoritmusok alapját az absztrakt domainként alkalmazott többdimenziós konvex poliéderek adják, így az implementációhoz nélkülözhetetlen egy olyan megoldó, amely segítségével ezek hatékonyan kezelhetők. Ilyen megoldó például a Parma Polihedra Library (PPL), amely pontosan ezt hivatott elősegíteni. A jelenlegi implementáció a poliédereken végzett műveletekhez a PPL megoldót veszi igénybe.

Az implementáció alapvető architektúrája (ld. az 5.1. ábra) a következő elemekből áll:

- *Bemenet:* bemenetként természetesen a modellt és a nemkívánt állapothalmazt várja az algoritmus. Az implementáció a TTMC nyelvén leírt modelleket illetve kényszereket tudja értelmezni.

- *Konfigurációs paraméterek:* szintén bemeneti paraméterek, azonban nem az elérhetőségi kérdésre, hanem az algoritmus futására vonatkoznak. Ilyen lehet pl. a terminálódást nem garantáló esetben a levágás

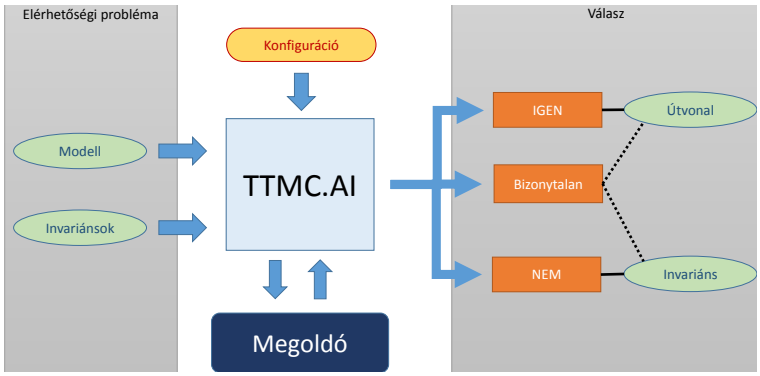
maximális száma, illetve az explicit állapoottérbejáró esetben a keresés mélysége.

- *Kimenet:* Az elérhetőségi analízis jelenlegi formájában tud biztos igent, biztos nemet, illetve bizonytalan választ visszaadni. Igen válasz esetén megadja az elérési utat is bizonyításként. Nem válasz esetén megadja a felülbecsült állapotteret. Bizonytalan válasz esetén megadja azokat az útvonalakat, amelyek folytatásán elvileg elérhetőnek tart olyan állapotokat, amelyekre teljesülnek a megadott kényszerek, illetve az utolsó explicit elérhető állapotból elérhető állapotokra vonatkozó invariánsokat.
- *TTMC.AI:* Az elérhetőségi analízist végző algoritmusokat tartalmazó komponens.
- *Mögöttes megoldó:* A poliéder műveleteket végző eszköz (jelenleg PPL). Futás során az algoritmus ennek a komponensnek a függvényeit hívja meg.

5.2. Validáció

Az algoritmus futása után a kapott eredmény, legyen az a megerősítő választ alátámasztó invariáns egyenlenlőtlenség rendszer, vagy a cáfoló választ alátámasztó elérési útvonal mindig ellenőrzhető. Ez alapján ellennőrizhető az algoritmus helyessége is.

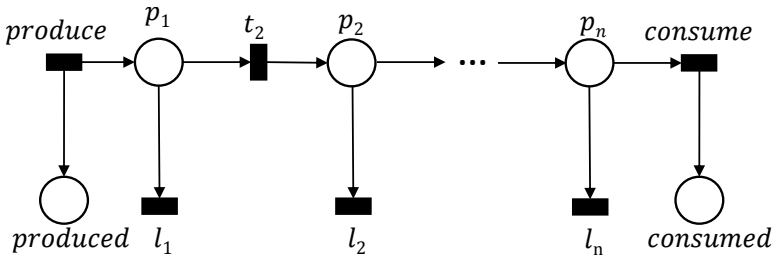
Az általam definiált algoritmus helyességellenőrzéséhez létrehoztam egyszerű teszteseket, amelyre ellenőriztem a válasz helyességét, valamint lefuttattam a szakirodalomban talált példákon, és a kapott eredményt összevetettem az ott leírtakkal. Az eredmények sosem mondtak ellent egymásnak, azonban sok esetben az én eszközöm precízebbnek bizonyult.



5.1. ábra. Achitektúra ábra

A mérési tapasztalatok alapján kijelenthető, hogy az algoritmus garantáltan lefut véges idő alatt, ráadásul a futásidő konfigurációs paraméterekkel közvetlenül vezérelhető. Futásidő tekintetében a tokenek mennyisége nem számít, de a struktúra bonyolodásánál (helyek, trnzíciók számának növekedése) várható komplexitásnövekedés.

A bemutatott algoritmus léptékezhetőségére teljesítménymméréseket végeztem. A mérés során a korábban bemutatott termelő-fogyasztó modell többlépéses variánsán az eredeti algoritmus által adott eredményen vizsgáltam azt a feltételezést, miszerint nem fogyasztódhatott el több elem mint amennyi termelődött. Az 5.2. ábrán látható a tesztesetek általános struktúrája. Így a 2.4. ábrán látható termelő-fogyasztó rendszer csatornája tetszőlegesen sok köztes hellyel bővíthető, melyek mindegyikén történhet tokenvesztés.

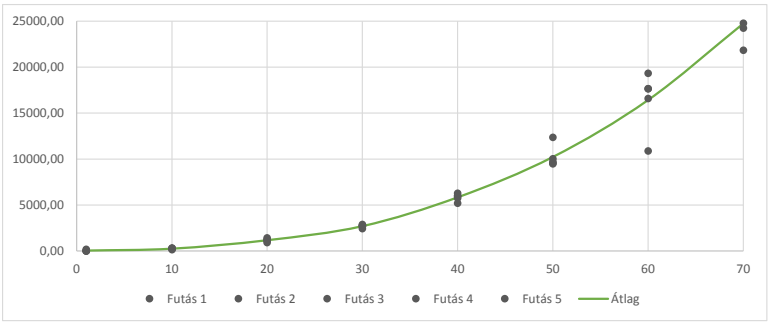


5.2. ábra. Mérés során vizsgált feladat.

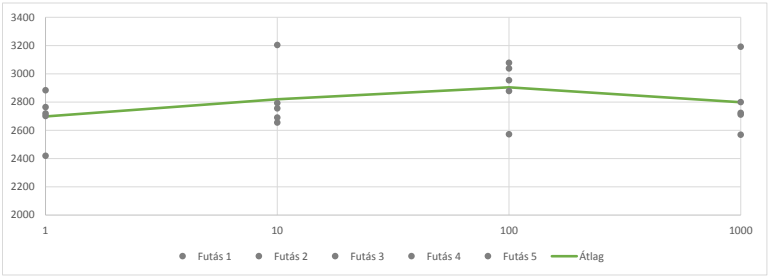
A mérési eredményeket az 5.3. ábrán látható diagram ábrázolja. A méréseket egy átlagos személyi számítógépen végeztem 4 GB memóriával i3-as 2 magos 2.2 GHz-es processzorral. Minden mérést ötször végeztem el, és a futási idők átlagát mértem milliszekundumban. A tesztesetekben a köztes csomópontok száma tízesével nőtt. Ezen a módon 70 köztes helyig sikerült mérni, utána memóriahiány miatt a program nem tudott lefutni. Egy mérést sikerült végezni 100 köztes hellyel rendelkező modellre, ekkor 70 másodperc alatt futott le az algoritmus.

Végeztem mérést 30 köztes hellyel, az élsúlyok változtatásával is. Ezzel azt a feltételezést vizsgáltam, hogy a tokenek száma nem befolyásolja az algoritmus futásidejét. A feltételezés helyességét az 5.4. ábra igazolja.

A mérési eredményeket kielégítőnek találom, megfelelő erőforrásokkal az analízis akár több száz elemből álló modelleken is lefut.



5.3. ábra. Mérési eredmények a modell méretének változtatásával



5.4. ábra. Mérési eredmények az élsúlyok változtatásával

6. fejezet

Összefoglalás és további lehetőségek

Ebben a fejezetben összefoglalom a munkám során elért, a dolgozatban bemutatott eredményeket, valamint ismertetem munkám további céljait.

6.1. Eredmények

A munkám során elért legjelentősebb kutatási eredmények a következő pontokban foglalhatók össze:

- A Petri-hálókra alkalmazott modellellenőrző algoritmust kiterjesztettem különböző Petri-háló alapú modellekre, többek között tiltó éleket tartalmazó Petri-hálókra és vezérlési állapotokkal rendelkező struktúrákra a modellező eszközök sajátosságainak figyelembevételével, valamint alkalmassá tettem az algoritmust általános Lineáris Tranzíciós Rendszerek kezelésére.
- Bevezettem egy saját megközelítést konvex poliéderek esetében alkalmazott fókuszálásra, mellyel az algoritmus által adott eredmény precíziója jelentősen növelhető.
- Létrehoztam több olyan algoritmust amelyek az elérhetőségi probléma célállapothalmazát felhasználva képesek finomítani az absztrakt állapottér-reprezentációt, így pontosabb választ adnak.
  - Petri-háló alapú modellekhez inverz működésű struktúrát származtattam, majd ennek felhasználásával definiáltam egy olyan algoritmust, amely a célállapot felől hátrafelé keres az állapottérben, ezáltal pontosabb eredményt adva.
  - Módszert adtam, amellyel az elérhetőségi kérdés megkötéseit felhasználva lehet új vezérlési állapotokat definiálni, és a modellt eszerint módosítani a pontosabb eredmény érdekében.

- Egységes keretrendszert javasoltam, amelyben több absztrakt interpretációs algoritmust együttesen lehet alkalmazni az eredmény precíziójának növelése érdekében.
  - Egyéb módszerek is tudnak invariánsokat szolgáltatni, melyeket a keretrendszer képes felhasználni.
  - Módszert adtam amely az absztrakt interpretációs algoritmust explicit állapottérbejárás alkalmazásával teszi hatékonyabbá, és ezt felhasználva képes útvonalat is adni a célállapothoz.
- Példákon demonstráltam, hogy a bevezetett módosítások alkalmazása tipikusan precízebb, de minden esetben legalább olyan precíz eredményt ad, mint az eredeti algoritmus.
- A javasolt keretrendszer egy kutatási prototípusát megvalósítottam, melyben a PPL matematikai megoldót modulként felhasználtam.
- Az elkészült keretrendszert sikerrel integráltam a tanszéken fejlesztett TTMC (Timed Transition systems Model Checker) keretrendszerbe.
- A prototípus eszköz erőforrásigényeit kezdeti mérési eredményekkel vizsgáltam.

6.2. További fejlesztési lehetőségek

Kutatásom elsődleges célja az algoritmus kiterjesztése időzített lineáris tranzíciós rendszer ellenőrzésére, mellyel iparilag releváns esettanulmányon is lehetővé válik az eszköz alkalmazása.

Implementáció területén a fő célom a TTMC ezközbe teljesen integrálni a TTMC.AI algoritmusát, hogy az a nyelven kívül a többi komponenssel (pl megoldók) is együttműködhessen, valamint a jövőben a TTMC részeként integrálható legyen a tanszéken jelenleg fejlesztett egységes absztrakt interpretációs keretrendszerrel.

Irodalomjegyzék

[1] Piotr Chrzastowski-Wachtel. Testing undecidability of the reachability in petri nets with the help of 10th hilbert problem. In *Application and Theory of Petri Nets 1999*, pages 268–281. Springer, 1999.

[2] Robert Clarisó, Enric Rodríguez-Carbonell, and Jordi Cortadella. Derivation of non-structural invariants of petri nets using abstract interpretation. In *Applications and Theory of Petri Nets 2005*, pages 188–207. Springer, 2005.

[3] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification*, pages 420–432. Springer, 2003.

[4] Michel Hack. The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 156–164. IEEE, 1974.

[5] Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[6] John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.

[7] Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[8] R.J. Lipton. *The Reachability Problem Requires Exponential Space.* Research report (Yale University. Dept. of Computer Science). Department of Computer Science, Yale University, 1976.

[9] Ernst W Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on computing*, 13(3):441–460, 1984.

[10] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[11] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Petri net analysis using invariant generation. In *Verification: Theory and Practice*, pages 682–701. Springer, 2003.

[12] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model Checking, and Abstract Interpretation*, pages 25–41. Springer, 2005.