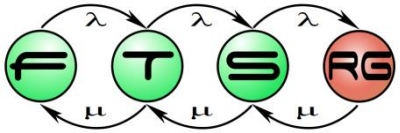
Petri-hálók elérhetőségi analízise ellenpélda-alapú absztrakció finomítással (CEGAR)

Hajdu Ákos
Mártonka Zoltán
Papp Pál András

Konzulensek:
Dr. Bartha Tamás (BME MIT)
Vörös András (BME MIT)



Bevezető

Alapfogalmak

- Elérhetőségi probléma
  - Adott célállapot elérhető-e a kezdőállapotból engedélyezett állapotátmeneteken keresztül
  - Eldönthető, de felső korlát nem ismert
  - Alsó korlát: legalább EXPSPACE nehéz
    - Exponenciális függvénnyel lehet leírni a tárhely igényt

Petri-háló elérhetőség

- Legtöbb eddigi megoldás elérhetőségi gráfon alapszik
- Elérhetőségi gráf:
  - kis háló esetén is lehet nagy
  - vagy akár végtelen
- Végtelen esetben a fedési gráfot tudjuk használni
  - Absztrakció
  - Véges reprezentáció

Petri-háló analízis

- Eldönthetetlen problémák:
  - 2 Petri-háló esetén az egyik elérhető állapotainak halmaza részhalmaza-e a másik háló elérhető állapotainak (subset problem)
  - 2 Petri-háló esetén az elérhető állapotok halmaza megegyezik-e (equality problem)
- Fedési probléma:
  - EXPSPACE nehéz

Alapfogalmak

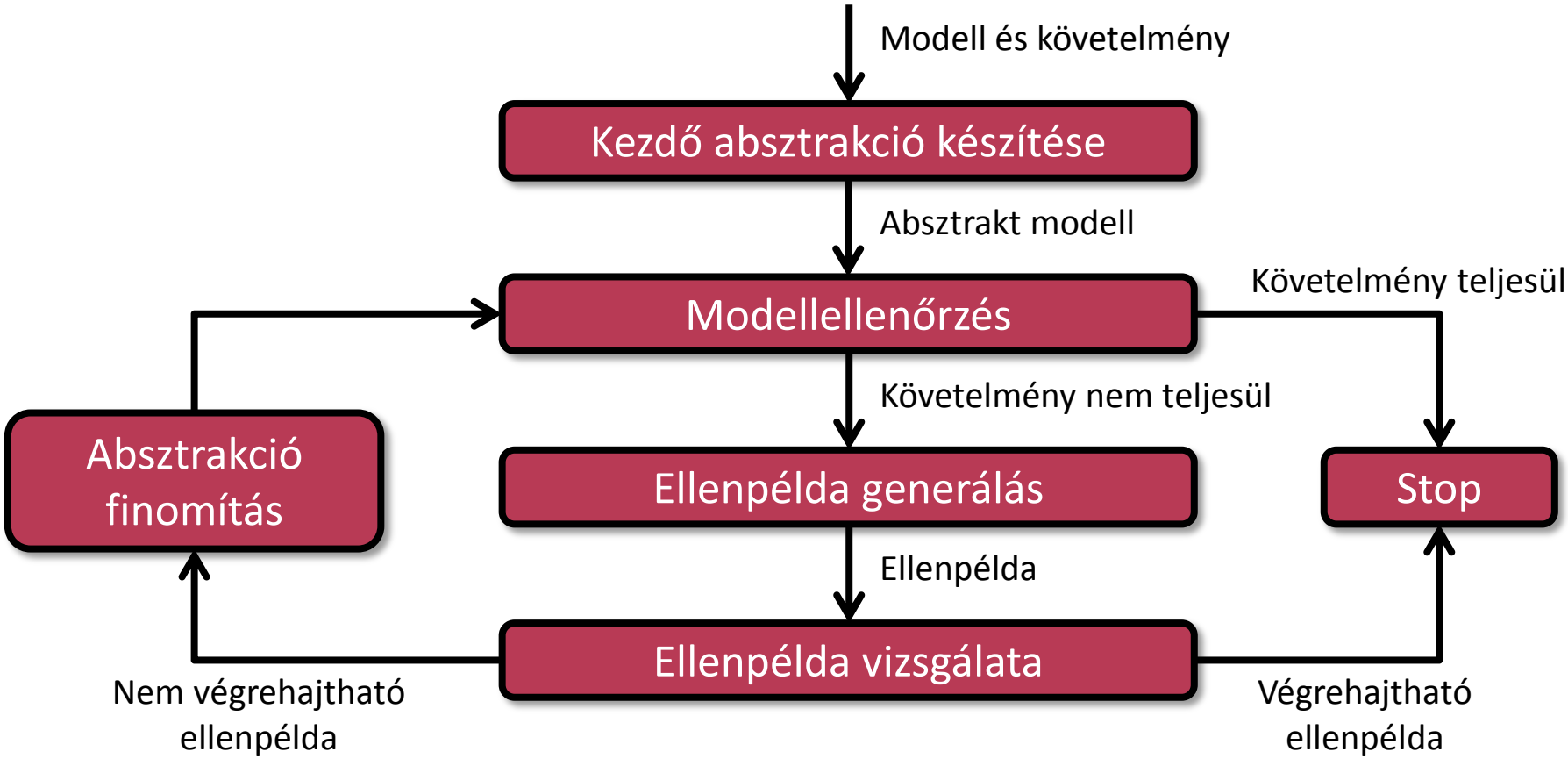
- Rész-elérhetőségi probléma
  - Lineáris feltétel(ek) a tokeneloszlásra vonatkozóan
  - Elérhető-e a feltételeket teljesítő állapot
- T-invariáns
  - Eltüzelése nem változtatja meg a tokeneloszlást

CEGAR megközelítés

- *(Counterexample guided abstraction refinement)*
- Ellenpélda-alapú absztrakció finomítás
  - Általános megközelítés végtelen állapotterű modellek vizsgálatára
  - Állapotok számát igyekszik csökkenteni, vagy véges absztrakciót adni
  - Absztrakt modell felülbecsli az eredetit
    - Absztrakt modell többféle viselkedéssel rendelkezik

CEGAR megközelítés

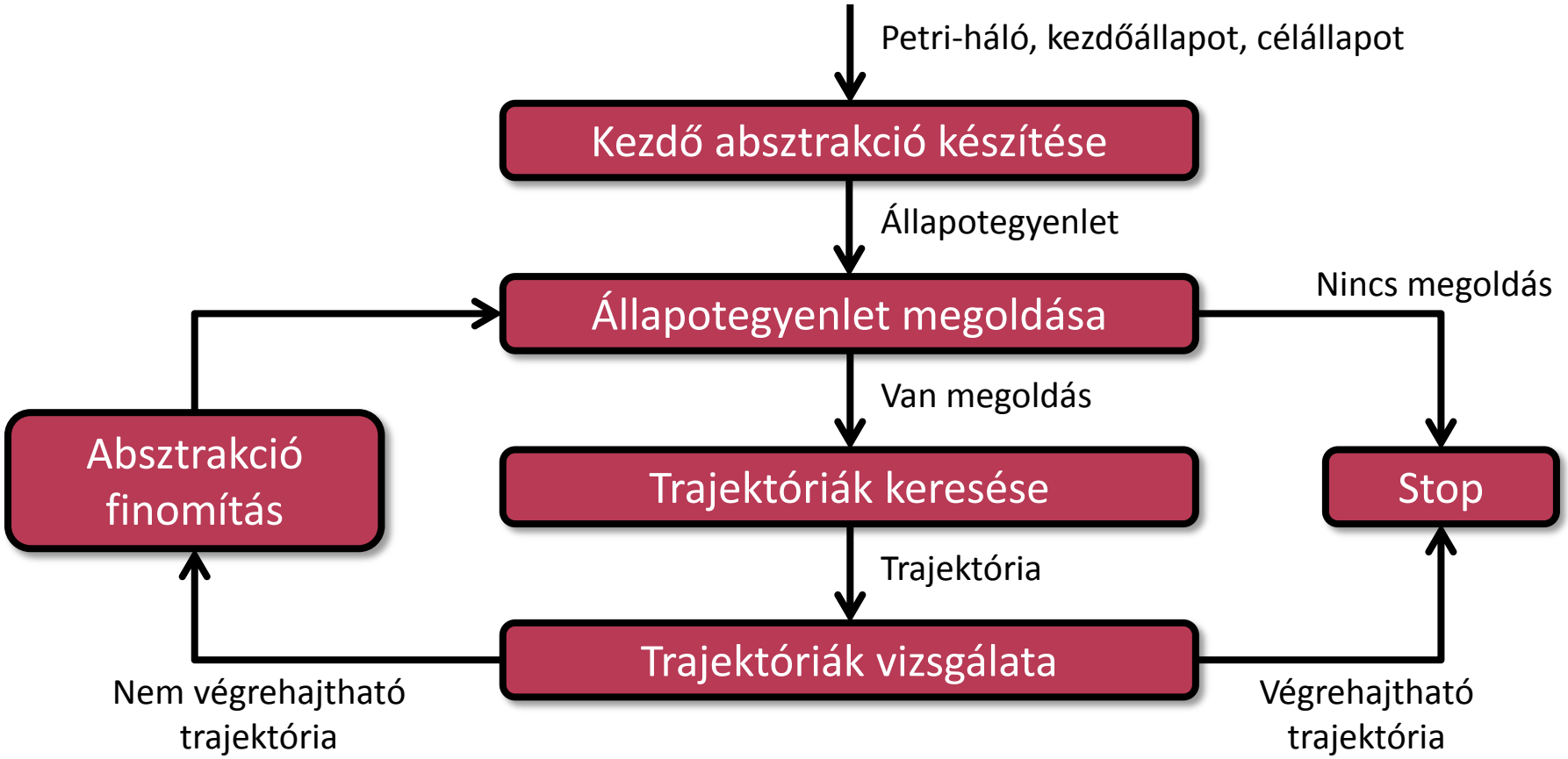
- Folyamatábra:



Petri-háló CEGAR algoritmus

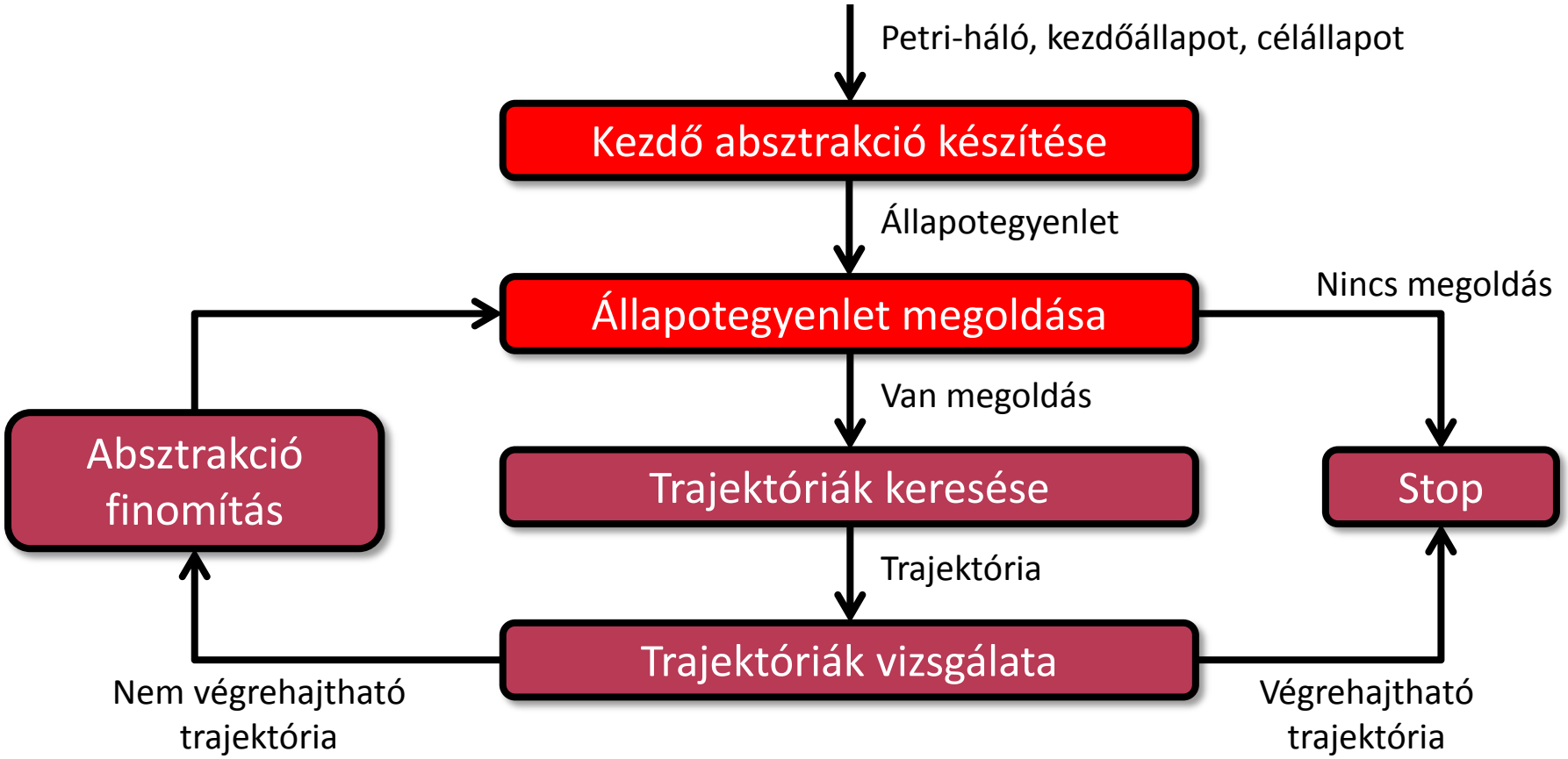
Petri-háló CEGAR algoritmus

- Folyamatábra:



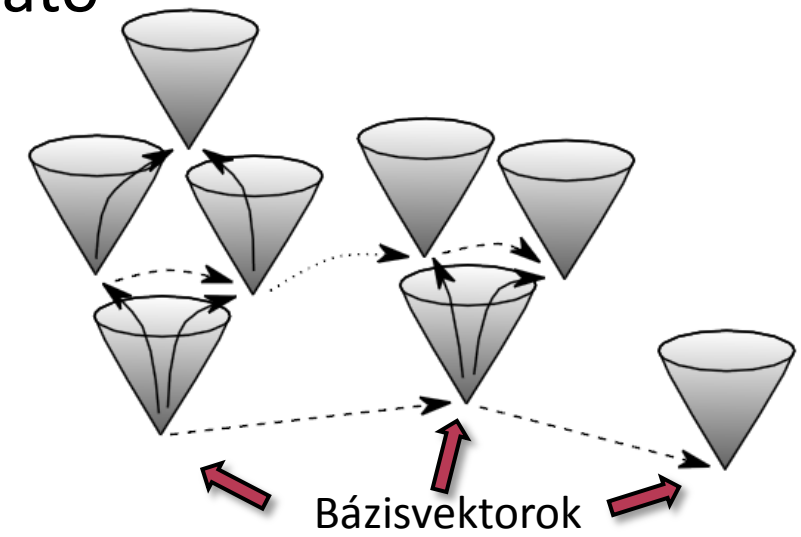
Petri-háló CEGAR algoritmus

- Folyamatábra:



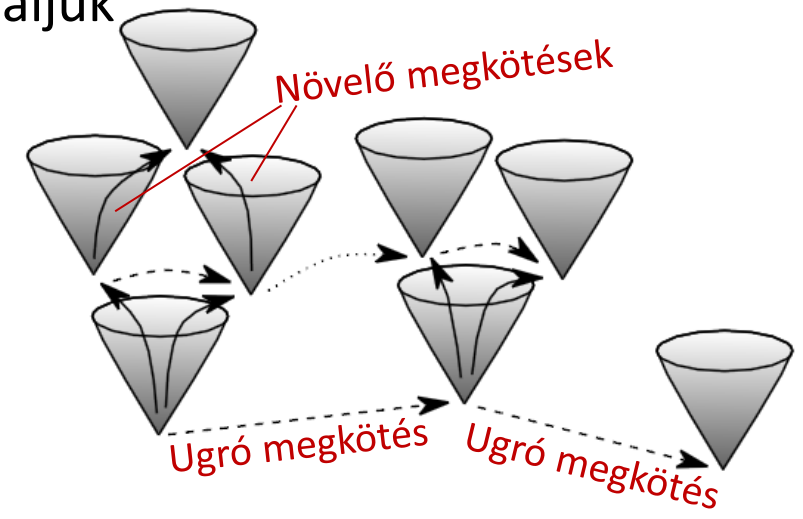
Absztrakció

- Absztrakció: állapotegyenlet
  - (Egészértékű) lineáris egyenletrendszer (ILP probléma)
  - Megoldhatósága szükséges, de nem elégséges feltétele az elérhetőségnek → jó absztrakció
- Állapotegyenlet tere
  - Minden megoldásvektor felírható bázisvektor és T-invariánsok lineáris kombinációjának összegeként



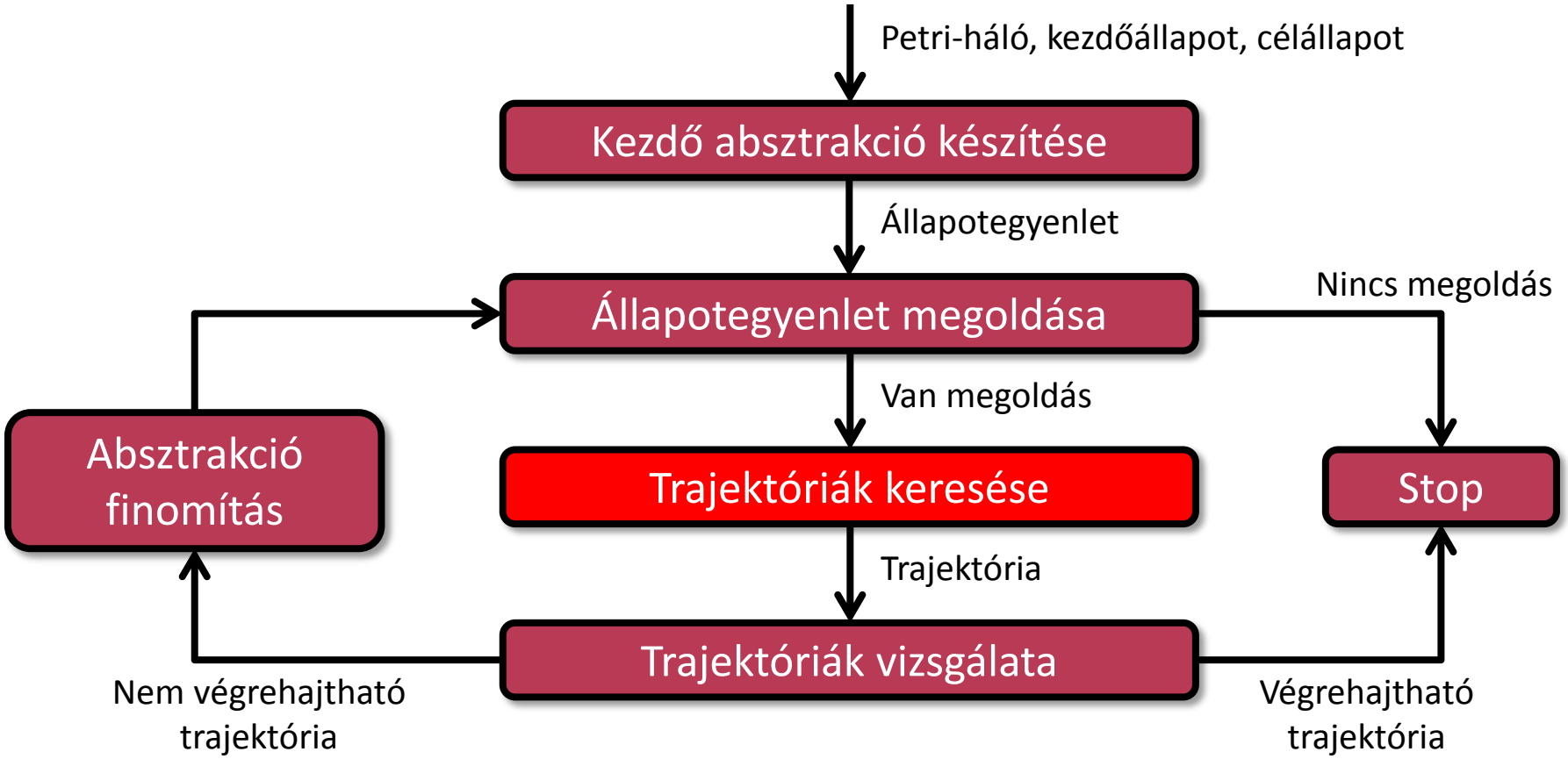
Absztrakció

- Állapottér bejárása irányítottan
  - Az ILP eszköz mindig egy megoldást produkál
    - Adott célfüggvény szerinti minimálisat
    - Jó célfüggvény: tranzíciók tüzelési számainak összege
  - Más megoldások kikényszerítése
    - Lineáris egyenlőtlenségek (megkötések) hozzáadása az állapotegyenlethez
    - Absztrakció finomítása során használjuk
  - 2 féle megkötés
    (később részletesen lesz róluk szó)
    - Ugró
      - Váltás bázisvektorok között
    - Növelő
      - Nem minimális megoldások elérése



Petri-háló CEGAR algoritmus

- Folyamatábra:



Trajektóriák keresése

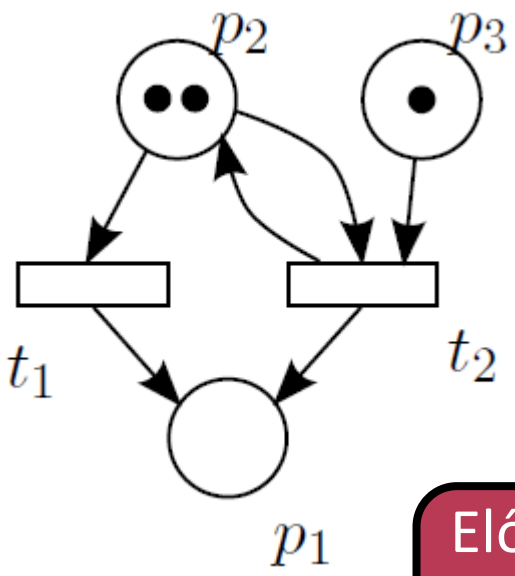
- Egy megoldásvektorhoz *részleges megoldás*ok rendelhetők úgy, hogy annyi tranzíciót tüzelünk el a megoldásvektorból, amennyit csak lehet
- Formálisan egy részleges megoldás 4 elemből áll:
  - Megkötéslista
    - Az állapotegyenletet a megkötéslistával kiegészítve kaptuk meg azt a megoldásvektort, ami alapján ezt a részleges megoldást generáltuk
  - Megoldásvektor
    - Az állapotegyenletet és a megkötéseket is kielégítő, adott célfüggvény szerinti minimális megoldásvektor
    - A megkötéslista és az állapotegyenlet alapján számítható, kényelmi szempontból tároljuk
  - Tüzelési sorozat
    - Maximális végrehajtható tüzelési sorozat
    - Minden tranzíció legfeljebb annyiszor szerepel, ahányszor a megoldásvektorban
  - Maradékvektor
    - Melyik tranzíciót hányszor kellene még eltüzelni

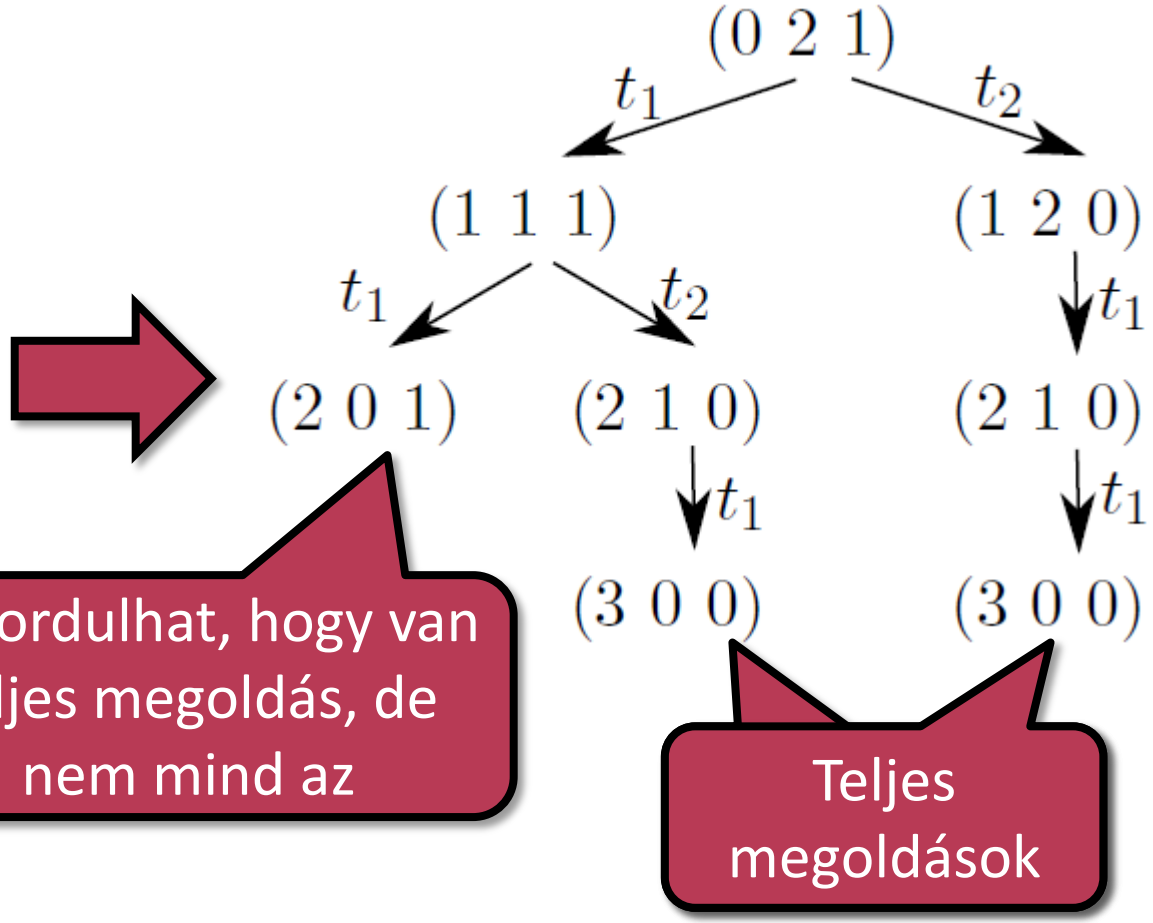
Trajektóriák keresése

- Részleges megoldások generálása
  - Fa építés
    - Csúcsok tokeneloszlások, élek tranzíciók tüzelései
    - Gyökérelem: kezdő tokeneloszlás
    - Leszármazottak a tranzíciók tüzelésével kaphatók
    - A gyökérből a levélbe vezető úton minden tranzíció legfeljebb annyiszor szerepelhet, ahányszor a megoldásvektorban szerepel
    - Maximális: levelekben nem lehet tüzelhető tranzíció
    - „Brute-force”, optimalizációkról később lesz szó
  - Minden levélből gyökérbe vezető út kijelöl egy tüzelési sorozatot, egyúttal egy részleges megoldást

Trajektóriák keresése

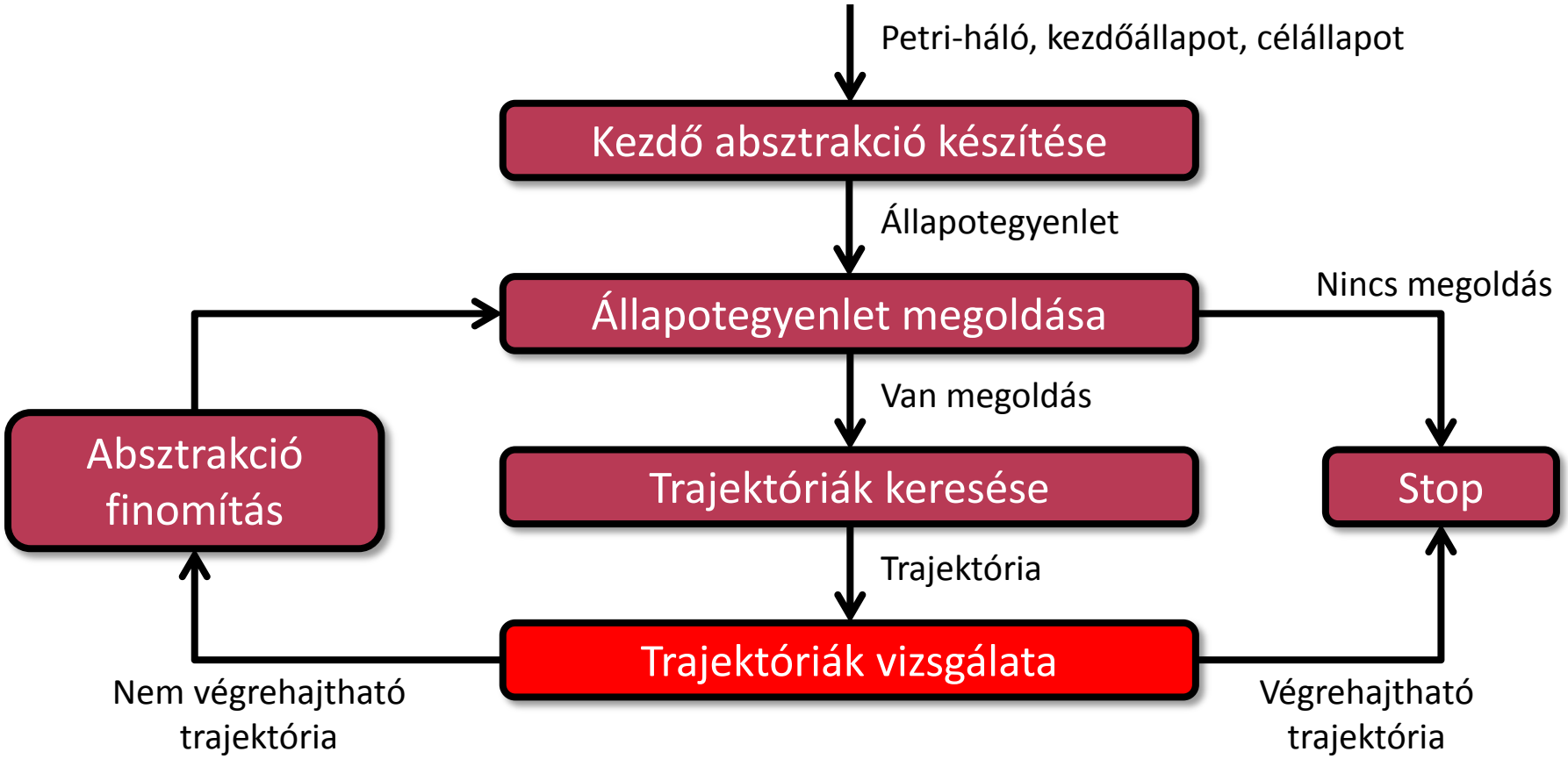
- Fa építés példa
  - Megoldásvektor: $[t_1\ t_2]=[2\ 1]$





Petri-háló CEGAR algoritmus

- Folyamatábra:

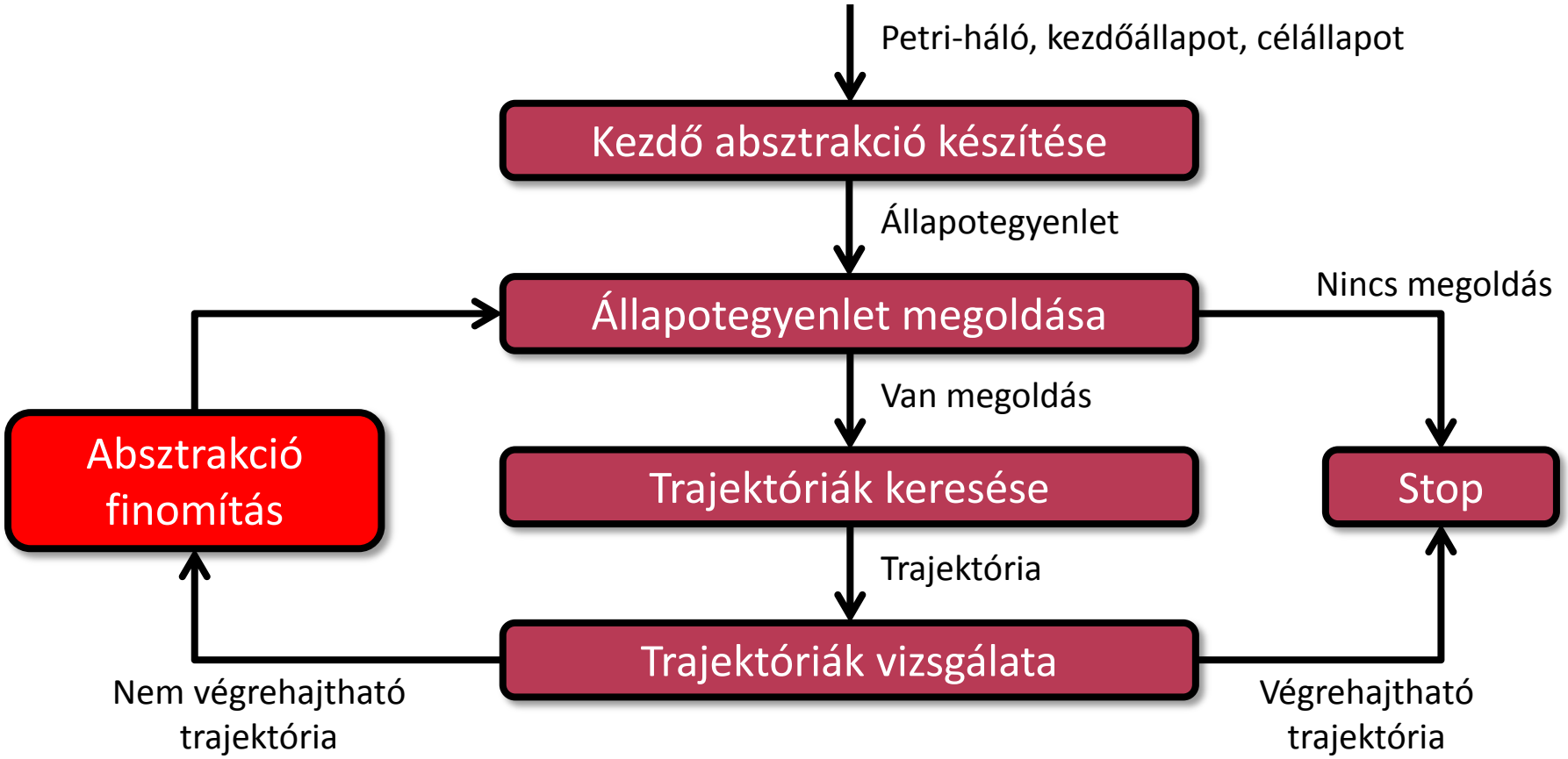


Trajektóriák vizsgálata

- Ha egy részleges megoldás maradékvektora nullvektor, akkor teljes megoldásnak nevezzük
  - Elégséges feltétel az elérhetőségre
- *Fontos: megoldásvektor != részleges megoldás*
  - *Megoldásvektor: állapotegyenlet egy megoldása*
  - *Részleges megoldás: megoldásvektor + egy lehetséges maximális tüzelési sorozat*
- Minden megoldásvektorhoz tartozik legalább egy részleges megoldás
  - Ha egy tranzíció sem tüzelhető, az is egy részleges megoldás
- Mi van akkor, ha van megoldásvektorunk, de nincs hozzá teljes megoldás?
  - Ez egy ellenpélda → absztrakció finomítás

Petri-háló CEGAR algoritmus

- Folyamatábra:

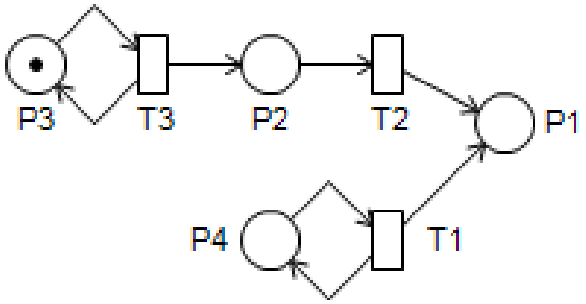


Absztrakció finomítás

- Adott egy megoldásvektor és hozzá tartozó részleges megoldások, de nincs teljes megoldás
  - Absztrakció finomítására van szükség
- 2 megközelítés
  - Megoldásvektor alapján
    - Ugró megkötéssel másik bázisvektort kaphatunk …
    - … amihez aztán újból kereshetünk trajektóriákat
  - Részleges megoldások alapján
    - Ha egy részleges megoldás nem teljes, akkor bizonyos tranzíciók nem tudtak elégszer eltüzelni
    - Növelő megkötéssel próbálunk egy T-invariánst hozzávenni
      - Ez ugyan a végállapoton nem változtat
      - De a tüzelése során „kölcsönadhat” tokeneket azokra a helyekre, amelyek miatt nem tudnak az előbbi tranzíciók tüzelni

Ugró megkötések

- Ugró megkötés
  - Alakja: $t_i < n$
    - Jelentése: Olyan megoldásvektort keresünk, ahol az i. tranzíció kevesebbszer tüzel mint n
  - Segítségével másik bázisvektort kaphatunk meg
    - Kihasználjuk, hogy a bázisvektorok páronként össze nem hasonlíthatók, pl. [1 0 0] és [0 2 0]
- Példa



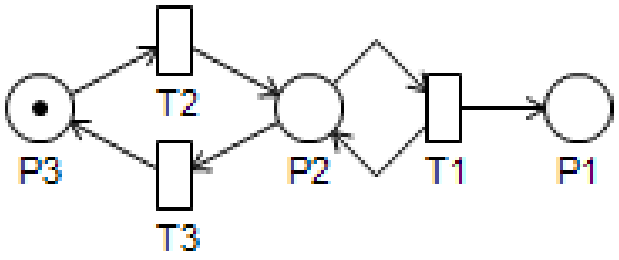
Elérhetőségi probléma: [0 0 1 0] -> [1 0 1 0]
Minimális megoldás: [1 0 0], azaz T1 eltüzelése
-> Nem realizálható
T1<1 megkötést hozzáadva új bázisvektor: [0 1 1]
-> T3, T2 sorrendben teljes megoldás

Ugró megkötések

- Ugró megkötések generálása
  - Megoldásvektor alapján készíthetők
  - Minden 0-nál többször tüzelő tranzícióhoz felvehető egy ugró megkötés
    - Példa: [1 2 0] megoldásvektor esetén két irányba mehetünk:
      - T1<1
      - T2<2
  - Ha kapunk újabb megoldásvektort arra meg kell ismételni az eljárást
    - Példa: tfh. a fenti példában T2<2 után kapunk egy [1 1 1] megoldásvektort, ekkor megint több irányba mehetünk, pl:
      - T1<1
      - T2<1
      - T3<1

Növelő megkötések

- Növelő megkötés
  - Alakja: $\Sigma n_i t_i \geq n$
    - Jelentése: Adott tranzíciók tüzelési számának súlyozott összege legalább n legyen
  - Segítségével nem minimális megoldásvektor érhető el
    - T-invariánsok lineáris kombinációját vesszük hozzá a bázisvektorhoz
- Példa



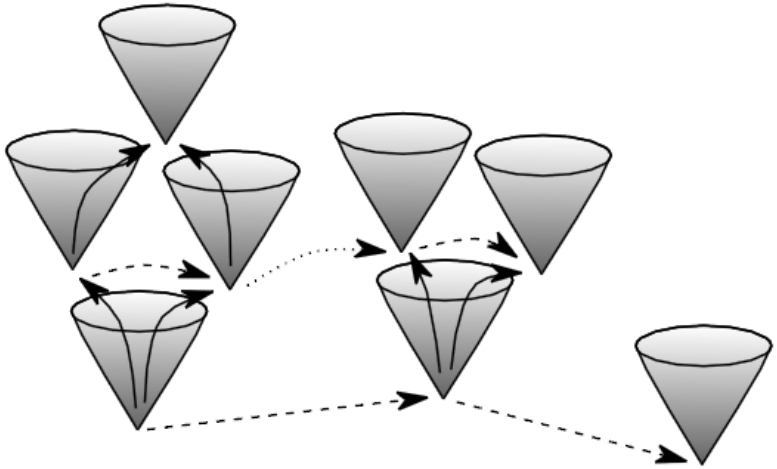
Elérhetőségi probléma: [0 0 1] -> [1 0 1]
Minimális megoldás: [1 0 0], azaz T1 tüzelése
-> Nem realizálható
T2>=1 megkötéssel nem minimális megoldás: [1 1 1]
-> T2, T1, T3 sorrendben teljes megoldás

Növelő megkötések

- Növelő megkötések generálása
  - Részleges megoldás alapján készíthetők
    - Ha egy részleges megoldás nem teljes, akkor bizonyos tranzíciók nem tudtak elégszer eltüzelni
    - Próbáljunk meg T-invariánsokkal az adott tranzíciók bemenő helyeire tokeneket „kölcsönözni”
    - Az algoritmus a részleges megoldás tüzelési sorozata utáni tokeneloszlás (végállapot) alapján dolgozik
  - 3 lépéses algoritmus
    1. Függőségi gráf építése és erősen összefüggő komponensek (SCC) keresése → kiderül mely helyekre kell token
    2. Szükséges tokenek számának megbecslése
    3. Helyekre vonatkozó feltétel átalakítása tranzíciók tüzelési számára vonatkozó feltétellé

Megkötések - összefoglalás

- Összefoglalás
  - Ugró megkötés
    - Megoldásvektor alapján
    - Másik bázisvektor elérése
  - Növelő megkötés
    - Részleges (nem teljes) megoldás alapján
    - Tokenek „kölcsönzése” T-invariánsokkal → nem minimális megoldásvektorok elérése
  - Keresés a megoldások terében
    - Növelő megkötésekkel mélységi jelleggel haladunk amíg tudunk
    - Ha elakadunk, új bázisvektorra ugrunk (szélességi jelleg)

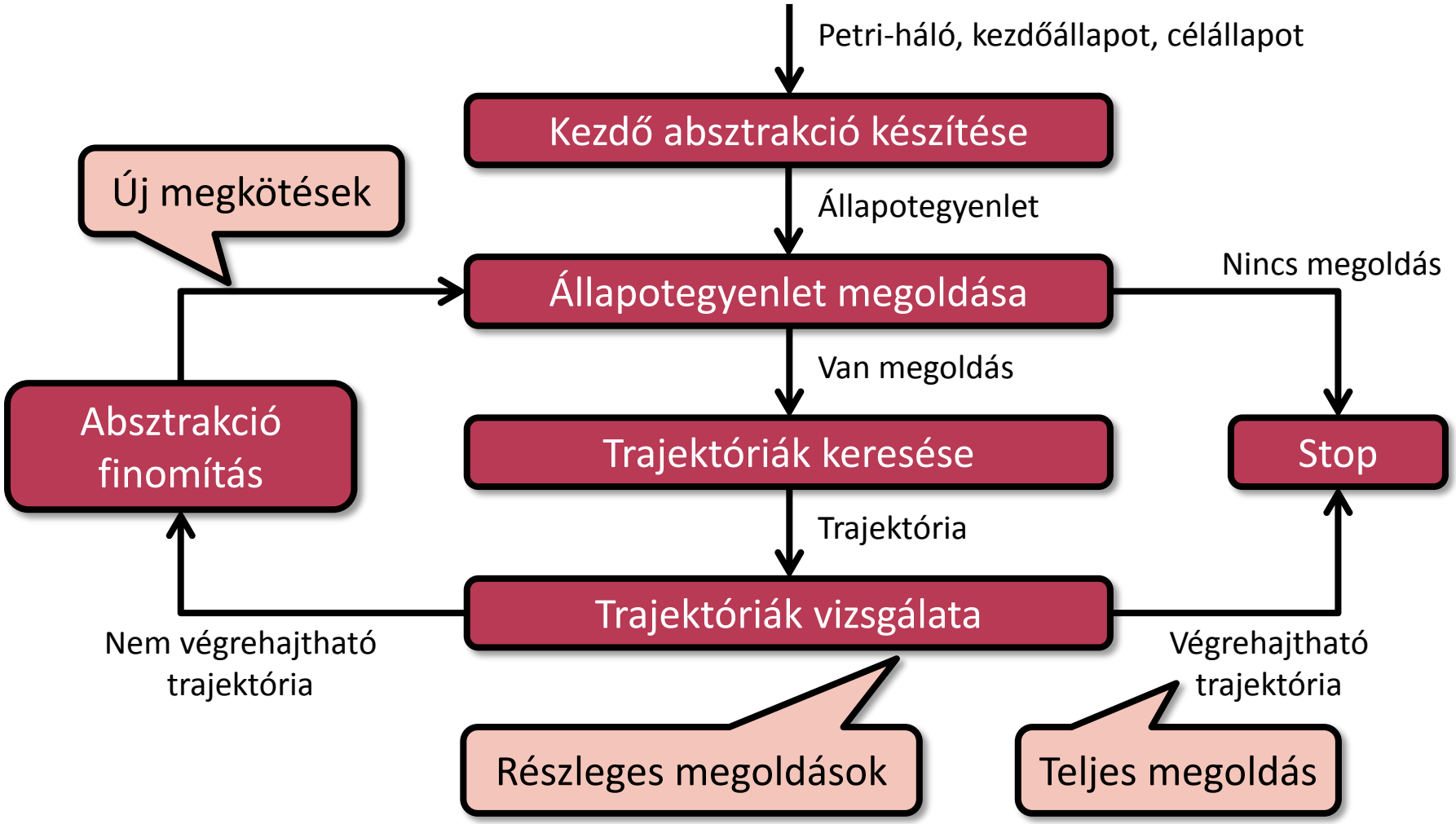


Megkötések - összefoglalás

- Ellentmondás a megkötések között
  - Ugró megkötések csak egy másik bázisvektor elérését szolgálják, utána áttranszformálhatók növelő megkötéssé
  - Példa: [1 1 0] megoldásból T1<1 megkötéssel megkapjuk [0 0 3]-at
  - Ezután ha pl. T1,T2 egy T-invariáns, akkor nem tudjuk hozzávenni a T1<1 ugró megkötés miatt
  - T1<1 valójában csak azt szolgálta, hogy [1 1 0] helyett [0 0 3]-at kapjunk, amit a T3>=3 növelő megkötéssel is elérhetünk
    - Így T1>=1 sem jelent már problémát

Petri-háló CEGAR algoritmus

- Hol tartunk?



Optimalizációk

Optimalizációk

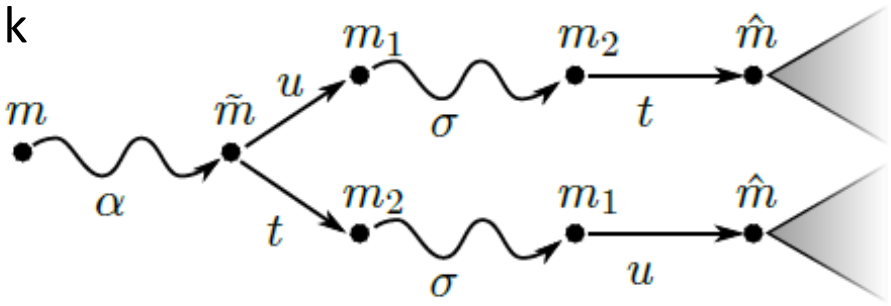
- Tartalom
  - „Brute force” faépítés optimalizálása
    - Stubborn set
    - Állapotalapú részleges rendezés
  - Keresési tér vágása
    - T-invairáns alapú szűrés
    - Szükséges feltétel teszt
    - Részleges megoldások tárolása

Stubborn set

- Részleges megoldások generálásakor a fa egy állapotából annyi irányban kell folytatni, ahány engedélyezett tranzíció van → „állapottér robbanás”
- Stubborn set módszer
  - Függőségek elemzésével csoportosítja a tranzíciókat
  - Minden állapothoz rendelhető egy ún. „stubborn set”
    - Stubbor set-be azok a tranzíciók kerülnek, amelyek irányában mindenképp folytatni kell a keresést
  - Stubborn set mérete kisebb vagy egyenlő, mint az összes engedélyezett tranzíció
- Extrém példa: hatványozó Petri-háló
  - Stubborn set nélkül 6 óra alatt sem futott le
  - Stubborn set-tel: ~ 8 mp

Állapotalapú részleges rendezés

- Amikor egy tranzíciónak többször is el kell tüzelnie, a stubborn set önmagában nem elég hatékony
  - Ugyanaz az állapot több, csak a tranzíciók sorrendjében különböző tüzelési sorozattal érhető el
    - Az alatta lévő részfát elég egyszer feldolgozni
    - Absztrakció finomítás során a tranzíciók sorrendjét úgysem vesszük figyelembe
  - Példa:
    - α tüzelési sorozat után (u σ t) és (t σ u) sorozatokkal is ugyanazt az állapotot érjük el
    - Ha a (u σ t) utáni részfát bejártuk a (t σ u) utánit már nem kell



T-invariáns alapú szűrés

- Adott egy részleges megoldás
  - Növelő megkötés hozzávesz egy T-invariánst
  - Kapunk egy új részleges megoldást, ami csak annyiban különbözik az előzőtől, hogy a T-invariánst eltüzeltük
    - Végállapot nem változik
    - Maradékvektor nem változik
    - (Látszólag) nem jutottunk előrébb
  - Az algoritmus ilyenkor ugyanazt az invariánst fogja hozzávenni újra → végtelen ciklust okozhat
    - A keresést ily módon nem folytathatjuk
    - Probléma: részleges megoldások elveszhetnek *(példa később)*

T-invariáns alapú szűrés

- Előfordulhat, hogy az invariánssal jó irányban haladtunk
  - Az invariáns tüzelése közben „kölcsönzött” valamennyi tokent a kívánt hely(ek)re
    - De nem eleget ahhoz, hogy egy maradékvektorban szereplő tranzíció engedélyezetté váljon
    - A részleges megoldás végállapotából ez nem derül ki, mert az invariáns teljes eltüzelése után a tokenek elkerültek a kívánt hely(ek)ről
  - Végig kell járni a tüzelési sorozatot és „jobb” köztes állapotokat keresni
    - „Jobb”: valamely maradékvektorban szereplő tranzíció bemenő helyéről összesen kevesebb token hiányzik, mint a végállapotban
    - „Jobb” köztes állapot egy új részleges megoldás → keresés folytatása

Szükséges feltétel teszt

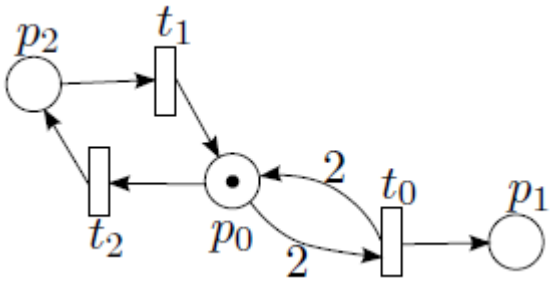
- Emlékeztető: növelő megkötés keresésének 3 lépése
  1. Függőségi gráf építése
  2. Tokenszám becslés
  3. Tranzíciók keresése

Itt már tudjuk, hogy mely helyekre és mennyi token szükséges

- A 2. és 3. lépés között az állapotegyenlet segítségével megvizsgáljuk, hogy az adott helyekre termelhető-e adott mennyiségű token (szükséges, de nem elégséges feltétel)
  - Ha nem teljesül a feltétel
    - Nincs szükség a 3. lépésre
    - Egyből elvágjuk a keresési teret (különben lehet, hogy csak több lépés után akadnánk el)

Szükséges feltétel teszt

- Működése
  - $m_0 + Cx >= m_1$
    - Állapotegyenlet módosított alakja
    - m_1: az adott helyekre előírjuk a szükséges tokenszámot, a többi helyre 0-t
  - Saját fejlesztés, bizonyítottan helyes feltétel a keresési tér vágására
- Példa



Elérhetőségi probléma: [1 0 0] -> [1 1 0]
Minimális megoldás: [1 0 0], azazT0 tüzelése
-> Nem realizálható
-> P0-ba kell két token
Az algoritmus T1,T2 invariánst venné hozzá újra és újra
[1 0 0] + Cx >= [2 0 0] nem oldható meg -> nincs megoldás

Részleges megoldások tárolása

- Megoldások tere átlapolódik
- Részleges megoldásokat eltároljuk, hogy ne dolgozzuk fel őket többször
- Kérdés: mikor ekvivalens két részleges megoldás?
  - Megkötések listája és a tüzelési sorozat egyértelműen meghatározza a részleges megoldást
    - Tüzelési sorozat helyett a megoldásvektor és a maradékvektor is használható (sorrend nem számít)
  - Megkötések listája mikor azonos?
    - Általuk meghatározott tér ugyanaz
    - Jelenlegi implementációban teljes egyezőség
      - Fölösleges megkötések ki vannak szűrve

Optimalizációk - összefoglalás

- Összefoglalás
  - Stubborn set és állapotalapú részleges rendezés
    - Részleges megoldások számának csökkentése
  - T-invariáns alapú szűrés
    - Végtelen ciklus megakadályozása
    - Megoldások elveszhetnek
  - Szükséges feltétel teszt
    - Bizonyítottan helyes vágási feltétel
  - Részleges megoldások tárolása

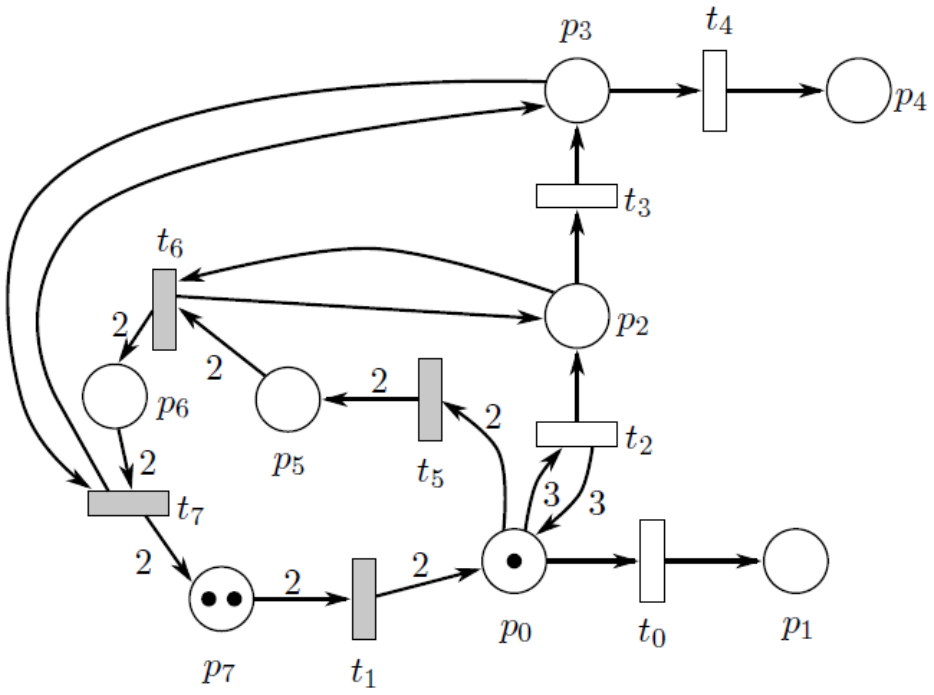
Algoritmus helyessége

Algoritmus helyessége

- Az algoritmusról bizonyították, hogy előállatja a végrehajtható megoldásokat
  - Bázisvektor + T-invariánsok lineáris kombinációjaként
- Növelő megkötés keresésekor használt becslő függvényről nem bizonyították, hogy helyes
  - A függvény a részleges megoldás tüzelési sorozata utáni állapotot (végállapot) használja fel
    - Nem feltétlenül ez a „legjobb” állapot, ami a tüzelési sorozatban előfordult
    - Ezt kihasználva ellenpélda konstruálható, ahol az algoritmus túlbecsli a szükséges tokenszámot, így nem talál megoldást

Algoritmus helyessége

- Ellenpélda (röviden):
  - Elérhetőségi probléma
    - Token P0-ból P1-be, illetve token P4-be
  - Megoldás
    - Két token P0-ba, T2 tüzelhető, így P4-be tud menni a token, majd T0 tüzelése
  - Algoritmus működése
    - T2-t nem tudja eltüzelni egyből, de T0-t igen
    - Emiatt P0-ban nem lesz token, így a szürkével jelölt invariánst kétszer hozzáveszi (pedig elég lenne csak egyszer)
    - Az invariáns úgy lett megkonstruálva, hogy minden tüzelése P4 felé visz egy tokent → csak egyszer tüzelhető



Algoritmus helyessége

- Probléma oka
  - Végállapot nem a „legjobb” bizonyos helyek szempontjából, így az ottani becslés rossz lehet
- Megoldás
  - Tüzelési sorozatban visszalépkedve érzékelni kell a jobb állapotokat
  - A jobb állapotokból való folytatás miatt sokfelé ágazna a keresés
    - Nem folytatjuk minden jobb állapotból
    - Egyszerűen ha volt jobb állapot, indítunk egy új ágat 1-es becsléssel
  - Az állapotalapú részleges rendezés optimalizáció kiszűrhet jobb köztes állapotokat → jelenleg is fejlesztés alatt

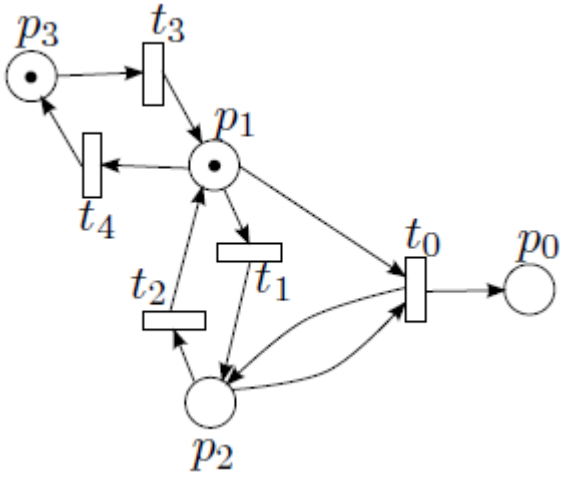
Algoritmus teljessége

Algoritmus teljessége

- T-invariáns alapú szűrés vágja a keresési teret
- Elveszhetnek végrehajtható megoldások, mert:
  - Az algoritmus nem vesz észre minden jobb köztes állapotot
  - Állapotalapú részleges rendezés miatt elveszhetnek jobb köztes állapotok
  - Egyéb okok miatt is, később lesz róluk szó

Algoritmus teljessége - ellenpélda

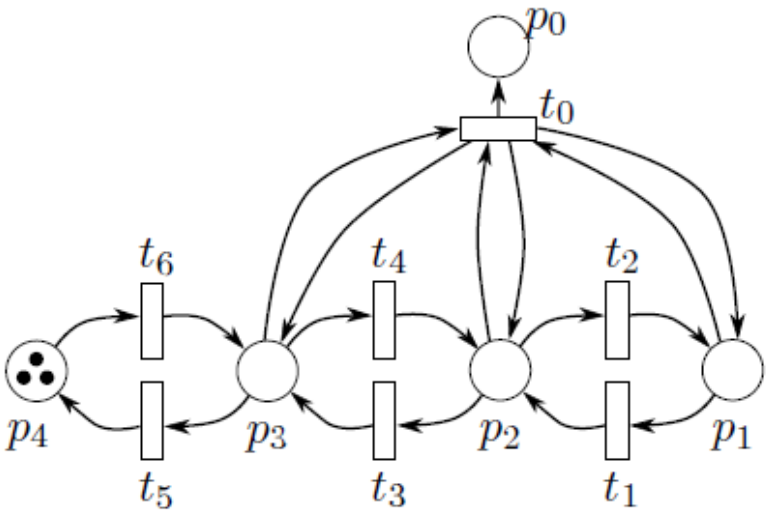
- Ellenpélda
  - Elérhetőségi probléma: [0 1 0 1] → [1 0 0 1]
  - Minimális megoldás T0 eltüzelése, ami P2 miatt nem megy
  - P2-re a T1-T2 invariáns tud tokent termelni, az algoritmus ezt veszi hozzá
  - T1-T2 eltüzelhető, de közben T0 nem válik engedélyezetté



  - Az algoritmus megint T1-T2 invariánst venné hozzá → kiszűri
  - Ha jobb köztes állapot alatt a tranzíció bemenő helyein a tokenek összegét értjük, akkor nincs jobb köztes állapot
    - T1 tüzelése után átkerül a token P2-be, de összesen ugyanúgy 1 token van T0 bemenő helyein, mint a végállapotban
  - Új definíció a jobb köztes állapotra: a tranzíció bármely bemenő helyén több token van, mint a végállapotban
    - Így T1 tüzelése után folytatva az algoritmus T3-T4 invariánst veszi hozzá és megoldja a problémát

Algoritmus teljessége - ellenpélda

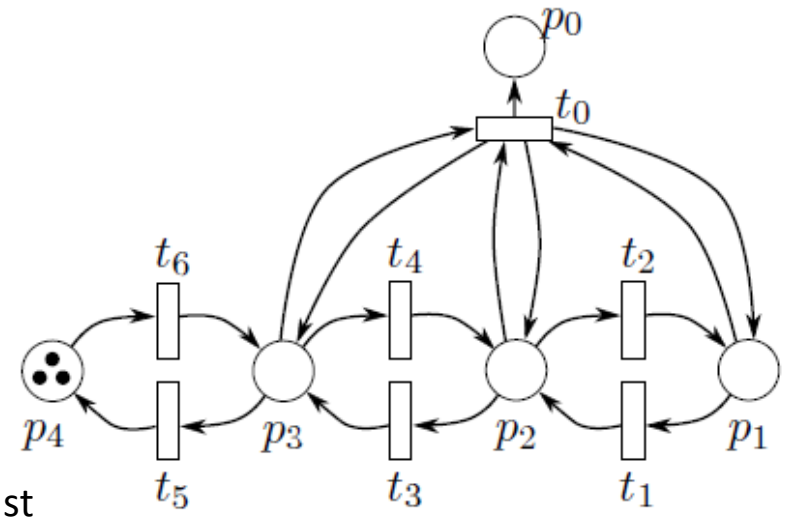
- Ellenpélda (vázlatosan)
  - Elérhetőségi probléma:
    [0 0 0 0 3] → [1 0 0 0 3]
  - Minimális megoldás T0 eltüzelése, nem realizálható
  - A megoldáshoz P4-ből a 3 tokent P1,P2,P3-ra kellene vinni



  - Az algoritmus először csak egy tokent hoz be, ettől T0 még nem tüzelhet, de van jobb köztes állapot (1 token P1,P2,P3 helyeken)
  - Ennek hatására behoz még egy tokent

Algoritmus teljessége - ellenpélda

- Ellenpélda (folytatás)
  - Tranzíciók tüzelési sorrendjétől függően két lehetséges forgatókönyv
    - Az egyik token bekerül P1,P2,P3 helyekre, majd ki is kerül, ezután a másodikkal ugyanez történik
      - Ilyenkor nincs jobb köztes állapot, mert a két token külön-külön járta be az invariánst
    - A két token egyszerre kerül be P1,P2,P3 helyekre
      - Van jobb köztes állapot



  - Állapotalapú részleges rendezés miatt előfordulhat, hogy az utóbbi tüzelési nem vesszük figyelembe
  - T-invariáns alapú szűrésnél újra fel kell építeni a részleges megoldás fáját, részleges rendezés nélkül
  - Lassít, de nélküle sok esetben nem találnánk meg a megoldást

Algoritmus teljessége - összefoglalás

- Problémák a teljességgel
  - Jobb köztes állapot szigorú definíciója
    - Megoldás: új sorrendezés
    - Ha bármely helyen több token van, az már jobb
  - Köztes állapotok elveszhetnek
    - Megoldás: fa felépítése újra
- Emlékeztető
  - Szükséges feltétel teszt → bizonyítottan helyes vágás
  - Bizonyos esetekben a T-invariáns alapú szűrés helyett használható, előnye:
    - Ha nem találunk megoldást, T-invariáns alapú szűrést használva nem tudjuk azt mondani, hogy tényleg nincs
    - Szükséges feltétel tesztet használva bizonyítottan nincs is megoldás

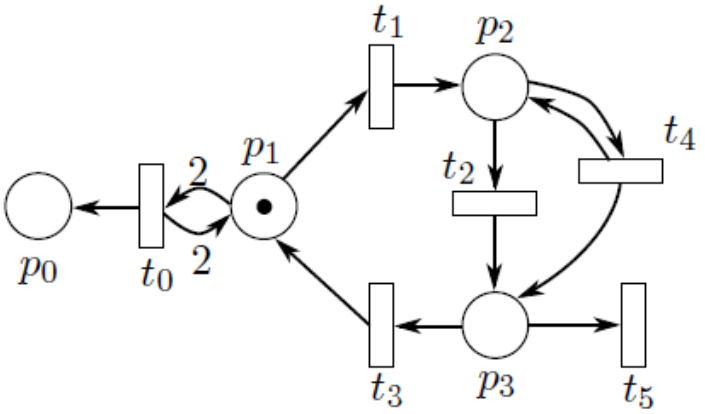
Algoritmus teljessége – további esetek

Algoritmus teljessége – további esetek

- Még mindig a T-invariáns alapú szűrés
  - Lehet-e olyan, hogy nem veszik el köztes állapot, nincs jobb köztes állapot és mégis van megoldás? → igen
- Oka: az algoritmus helyekben gondolkozik
  - Tüzelni nem tudó tranzíciók bemenő helyeire közvetlenül próbál tokent termelni
- Problémák:
  - Közvetett token termelés
  - Előbb el kell venni tokeneket helyekről, hogy továbbiakat termelhessünk

Algoritmus teljessége – további esetek

- Ellenpélda
  - Elérhetőségi probléma:
    [0 1 0 0] → [1 1 0 0]
  - Minimális megoldás: T0 tüzelése,
    P1 miatt nem realizálható



  - P1-re a T1-T2-T3 invariáns tud tokent termelni
    - Ezen az invariánson azonban ugyanaz a token megy körbe-körbe
    - Nincs jobb köztes állapot
  - Probléma:
    - P1-re közvetlenül nem tudunk több tokent termelni
    - De P3-ra igen, T4 segítségével (ehhez a már meglévő tokent ráadásul el is kell venni P1-ről és P2-re rakni)
    - A P3-ra termelt token a T1-T2-T3 invariánssal eljuttatható P1-re

Algoritmus teljessége – új megközelítés

- Új megközelítés (saját fejlesztés)
  - Kiszűrt T-invariáns kezelése egy egységként
    - Azért lett kiszűrve, mert nem tudott az adott hely(ek)en segíteni
    - Nincs rajta elég token
  - Próbáljunk meg tokent tenni rá
    - Majd közvetve eljuttatható a kérdéses hely(ek)re
  - Megoldandó problémák
    - Szükséges tokenszám
    - Terminálási feltétel

Algoritmus teljessége – új megközelítés

- Megoldandó problémák
  - Szükséges tokenszám → 2 lehetőség:
    - Becslés
      - Nem triviális
      - Tokenek számolása nehéz
      - Köztes állapotok itt is számítanának
    - Inkrementális módszer
      - Tegyünk 1 tokent az invariánsra
      - Ha nem lenne elég, akkor ezt ismételjük
      - Jelenlegi implementációban ezt alkalmazzuk

Algoritmus teljessége – új megközelítés

- Megoldandó problémák
  - Terminálási feltétel
    - Ha mindig minden kiszűrt invariánsra tokent termelnénk az végtelen ciklust okozhatna
  - Megoldás
    - Kiszűrt megoldás és T-invariáns (pl.: T1) megjelölése
    - Ha egy leszármazottja újból kiszűrődik miatta (pl.: T2 invariáns miatt), akkor:
      - Ha volt jobb köztes állapot, akkor újból T1-re kell tokent termelni
      - Ha nem volt jobb köztes állapot, de T2 nem részhalmaza T1-nek, akkor (T1 U T2)-re kell tokent termelni, így működik az is, ha egy invariánsra is közvetetten lehet csak tokent termelni, de kiszűri, ha pl.: T1 és T2 egymásra próbálna tokent termelni
      - Egyéb esetben nem kell folytatni

Ez most invariánst jelöl, nem tranzíciót

Algoritmus kiterjesztése új problémaosztályok kezelésére

Rész-elérhetőségi probléma és tiltó éles hálók

Algoritmus kiterjesztése: rész-elérhetőség

- Rész-elérhetőségi probléma
  - Lineáris feltételek a tokeneloszlásra vonatkozóan, például:
    - *3*P1 > 5* és *2*P1 + P3 = 7*
  - Az ilyen feltételek felírhatók mátrixos alakban:
    - $Am \geq b$, ahol:
      - A: együtthatómátrix
      - m: elérendő állapot
      - b: együtthatóvektor
  - Megoldandó probléma:
    - Elérendő állapotra vonatkozó feltételt át kell alakítani tranzíciókra vonatkozó feltétellé, mert az ILP eszköz azt képes kezelni

Algoritmus kiterjesztése: rész-elérhetőség

- Feltétel átalakítása
  - Az $Am \geq b$ egyenlőtlenségben az elérendő m állapot helyére behelyettesítjük az állapotegyenletet: $(m_0 + Cx = m) \rightarrow A(m_0 + Cx) \geq b$
  - Átrendezve:
    - $(AC)x \geq (b - Am_0)$
  - Ez már odaadható az ILP eszköznek
- Megoldások tere
  - A megoldások tere ilyenkor kicsit más felépítésű
    - Egy növelő megkötés nem feltétlenül T-invariánst vesz hozzá, lehet, hogy csak egy részhalmazát

Algoritmus kiterjesztése: tiltó élek

- Tiltó élek kezelése
  - Ha egy tranzíció tiltó él miatt nem tud tüzelni, akkor az adott hely(ek)ről el kell venni a tokeneket
- Újfajta megkötés
  - Olyan mint a növelő megkötés, csak fordítva működik
  - 3 lépéses algoritmus az új megkötés generálására:
    - Mely helyekről kell elvenni a tokeneket
    - Hány tokent kell elvenni
    - Tranzíciók tüzeltetése, amelyek el tudják venni a tokeneket

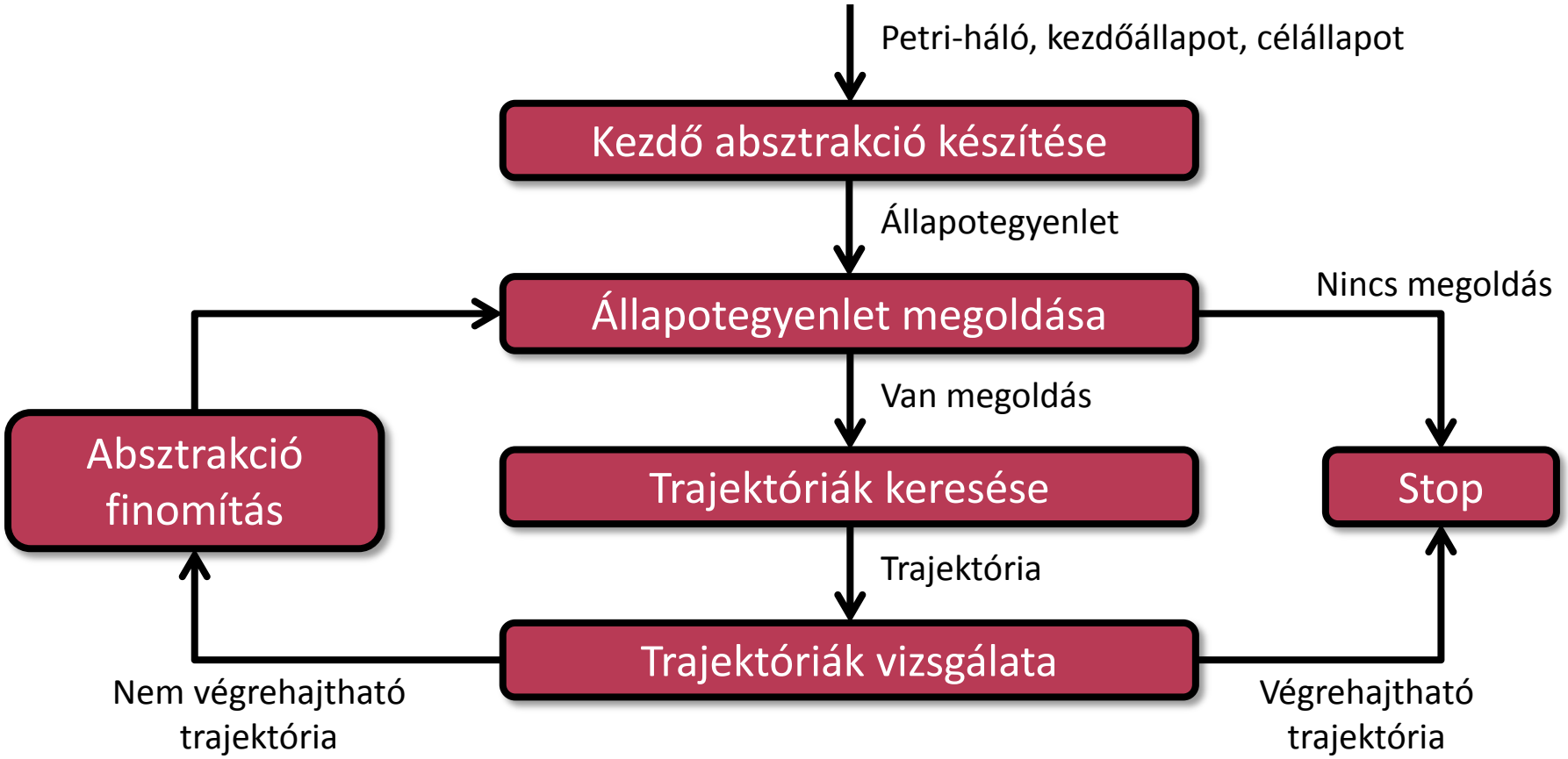
Algoritmus kiterjesztése: tiltó élek

- Optimalizációk
  - Stubborn set ilyenkor nem használható
  - Állapotalapú részleges rendezés ugyanúgy működik
  - T-invariáns alapú szűrés
    - Tiltó élekkel csatlakozó helyek esetén az a „jobb”, ha kevesebb a token mint a végállapotban
  - Szükséges feltétel teszt
    - Azt kell vizsgálni, hogy elvehető-e egyáltalán adott mennyiségű token
  - Részleges megoldások tárolása ugyanúgy működik
- Tiltó élekkel kiegészített hálók esetén az elérhetőségi probléma bizonyítottan nem eldönthető
  - Ennek ellenére sok esetben működik az algoritmus

Összefoglalás

Összefoglalás

- Petri-háló CEGAR algoritmus:



Összefoglalás

- Optimalizációk
  - „Brute force” faépítés optimalizálása
    - Stubborn set
    - Állapotalapú részleges rendezés
  - Keresési tér vágása
    - T-invairáns alapú szűrés
    - Szükséges feltétel teszt
    - Részleges megoldások tárolása

Összefoglalás

- Algoritmus helyessége
  - Probléma és javítása
- Algoritmus teljessége
  - Problémák és javításuk
  - További esetek
- Algoritmus kiterjesztése
  - Rész-elérhetőség, tiltó élek

Irodalom

- Harro Wimmel and Karsten Wolf: Applying cegar to the petri net state equation.
  CoRR, abs/1208.2159, 2012.
- Hajdu Ákos, Mártonka Zoltán: Diszkrét dinamikus rendszerek viselkedésének felderítése ellenpélda-alapú absztrakció finomítás (CEGAR) segítségével
  TDK dolgozat, 2012