metin, yazı tipi, logo, simge, sembol içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Nondeterministic Turing Makineleri (NDTM) Üzerine İnceleme**

**PROJE-4**

DOĞUKAN ÖZEK- 171422001

DUYGU İREM KILIÇ-170422027

EMRE KILIÇ- 170423512

**DERSİN ÖĞRETİM ÜYESİ**  
Dr. Öğr. Üyesi Gözde KARATAŞ BAYDOĞMUŞ

**LABORATUVAR SORUMLUSU**

Arş. Gör. Büşra BÜYÜKTANIR

İçindekiler

1. NDTM'nin Dil Karar Verme Yetenekleri
2. NDTM ve DTM Arasında Dil Karar Verme Eşdeğerliği
3. Uygulamalı Örnekler
4. Turing Makinesi Nedir
5. NDTM NEDİR
6. NDTM ve DTM Farkları
7. DTM VE NDTM Örneği
8. Halting Problemi
9. König Lemmaları
10. Yapı, Hesaplanabilirlik ve Seçim Aksiyomlarıyla İlişkisi

**Nondeterministic Turing Makineleri—Dil Karar Verme**

**1. NDTM'lerin Dil Karar Verme Yetenekleri**

Nondeterministic Turing Makineleri (NDTM), deterministik makinelerden (DTM) farklı olarak bir durumdan birden fazla farklı duruma aynı anda geçiş yapabilir. Bu özellik, NDTM'lerin bir dilin elemanı olup olmadığına karar verme sürecini etkiler. Bir NDTM'nin belirli bir dili karar verebilmesi için şu şartları sağlaması gerekir:

* **x ∈ L ise**: En az bir hesaplama yolu x'i kabul eder (accept), ve hiçbir yol x'i reddetmez (reject).
* **x ∉ L ise**: En az bir hesaplama yolu x'i reddeder ve hiçbir yol x'i kabul etmez.

Bu şartlar, NDTM'nin "karar verme" mekanizmasının temelini oluşturur. Ancak NDTM'ler, karar verme sürecinde genellikle "paralel" çalışan hesaplama yolları kullanır. Bir hesaplama yolunun kabul etmesi, tüm hesaplama sürecini kabul anlamına gelir. Bu bakış açısı, NDTM'leri belirli problemlerde DTM'lerden daha "esnek" yapar.

**2. NDTM ve DTM Arasında Dil Karar Verme Eşdeğerliği**

**Teorem:** Bir dil L, bir NDTM tarafından karar verilebiliyorsa, aynı zamanda bir DTM tarafından da karar verilebilir.

Bu teorem, DTM ve NDTM'lerin karar verme gücü açısından eşdeğer olduğunu ifade eder. Bunun anlamı, nondeterministik bir yapıya sahip olmasına rağmen, NDTM'lerin çözebildiği tüm dillerin deterministik yöntemlerle de çözülebilir olmasıdır.

**Kanıt:**

1. Bir NDTM'nin M, bir girdiyi x kabul ettiğini varsayalım. Bu durumda, M'nin en az bir hesaplama yolu kabul durumunda sonlanır.
   * Bu hesaplama yolu, nondeterministik bir yapının sonucu olsa da, tüm olası geçiş yolları kaydedilebilir ve tek tek simüle edilebilir.
2. Bir DTM oluşturularak bu hesaplama yolları tümüyle taranabilir. DTM, tüm olası durumları sistematik bir şekilde inceleyerek, x'in kabul edilip edilmediğine karar verebilir.
   * Bunun için DTM, her bir hesaplama yolunu derinlik-öncelikli (depth-first) veya genişlik-öncelikli (breadth-first) bir arama algoritmasıyla inceler.
   * Kabul eden bir yol bulunduğunda x kabul edilir; tüm yollar incelenir ve kabul eden yol bulunmazsa x reddedilir.
3. Bu yaklaşım, her NDTM için eşdeğer bir DTM oluşturulabileceğini gösterir. Dolayısıyla, karar verilebilirlik açısından NDTM ve DTM eşdeğerdir.

**3. Uygulamalı Örnekler**

**Örnek 1: Palindrom Dili**

* **Dil:** L₁ = {w | w, 0 ve 1'lerden oluşan bir palindromdur (tersi kendisine eşittir)}.
* **NDTM Çözümü:**
  + NDTM, girdinin ortasından başlayarak eşleşmeleri kontrol eder. Her olası bölünmeyi paralel olarak inceler.
  + Eğer herhangi bir paralel dalda eşleşme sağlanmazsa, bu dal reddeder. Tüm dallar reddederse girdi reddedilir.
  + En az bir dal kabul ederse, girdi kabul edilir.
* **DTM Çözümü:**
  + DTM, tüm olası bölünmeleri sistematik olarak tek tek inceler. Her eşleşmeyi sırayla kontrol eder ve tüm olasılıkları tüketir.
  + Paralel dalları simüle etmek yerine sıralı bir işlem yapar.

**Örnek 2: Dengeli Parantez Dili**

* **Dil:** L₂ = {w | w, doğru şekilde dengelenmiş parantez dizileridir}.
* **NDTM Çözümü:**
  + NDTM, her adımda bir açık parantezi kapalı bir parantezle eşleştirmek için tüm olasılıkları paralel olarak dener.
  + Eğer bir eşleşme eksik kalırsa veya kapalı parantezler açık olanlardan önce gelirse, bu dal reddedilir.
  + En az bir geçerli eşleşme bulunursa, girdi kabul edilir.
* **DTM Çözümü:**
  + DTM, bir yığın veri yapısı kullanarak her açık parantezi sıralı bir şekilde kapalı parantezlerle eşleştirir. Bu işlem sırasıyla ilerler ve tüm eşleşmeler doğruysa kabul edilir, aksi halde reddedilir.

**Örnek 3: Modüler Aritmetik Tabanlı Dil: 'a' Karakterlerinin Sayısı 2'nin veya 3'ün Katı**

* Dil: L = {w | w'deki 'a' karakterlerinin sayısı 2'nin katı veya 3'ün katıdır}
* **NDTM Çözümü:** 
  + Girdi paralel olarak iki yol kullanılarak incelenir. Birinci yol 'a' karakterlerinin sayısını 2'ye, ikinci yol ise 3'e göre kontrol eder. Her iki durumda da sayım yapılır ve eğer sayılar 2 veya 3'ün katıysa, kabul edilir.
* **DTM Çözümü:**
  + Girdi tek bir geçişte okunur ve bir sayaçla 'a' karakterlerinin sayısı sayılır. Sayaç, 2 ve 3'e göre kontrol edilir. Eğer sayaç birden fazla durumda (2 veya 3'ün katı) eşleşirse, kabul edilir. Aksi halde, reddedilir.
* **Detaylı Örnek: Düzensiz Dillerin Karar Verilmesi**
  + **Dil:** L = {w | w, "ab" veya "ba" substring’lerini içerir}.
  + **NDTM Yaklaşımı:**
    - Girdiyi tüm olası pozisyonlarda ikiye böler ve "ab" veya "ba" olup olmadığını paralel olarak kontrol eder.
  + **DTM Yaklaşımı:**
    - Her substring’i sıralı olarak inceler ve bu diziye rastlanıp rastlanmadığını kontrol eder.

-------------------------------------------------------------------------------------------------------------------------

1. **Turing Makinesi Nedir**

Turing Makinesi (TM), bilgisayar biliminin temelini oluşturan teorik bir modeldir. 1936 yılında **Alan Turing** tarafından tanımlanmıştır. Bu model, hesaplanabilirlik teorisinin temel taşını oluşturur ve modern bilgisayarların teorik bir soyutlaması olarak görülür.

Turing makinesinin çalışma prensibi teorik olarak sonsuz bir uzunlukta banta dayanır. Bant üzerinde hücreler bulunur ve bu hücreler bir sembol veya boş olabilir. Bant üzerinde bu hücreler arasında hareket eden bir bant başlığı vardır. Bu bant başlığı hücreler arasında hareket ederek sembolleri okuyabilir, yazabilir, sola veya sağa hareket edebilir.

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Örnek olarak

L={w#w | w ε {0,1}\* } (0 ve 1 lerden oluşan palindrom stringleri tanımlayacak bir dil) bu dili tanıyan bir Turing Makinesi örneği;

diyagram, metin, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **NDTM Nedir**

Non-deterministic Turing Machine (NDTM), klasik **Deterministic Turing Machine (DTM)** modelinin genişletilmiş bir versiyonudur. Bu model, bir Turing makinesinin **aynı anda birden fazla işlem yolunu takip edebilmesini** teorik olarak sağlar. Non-determinism (belirsizlik) sayesinde, NDTM, bazı problemleri çözmek için tüm olasılıkları **eşzamanlı** olarak dener.

diyagram, çizgi, kalıp, desen, düzen içeren bir resim

Açıklama otomatik olarak oluşturuldu

NDTM’ler DTM’lere göre bir stateten başka bir state’e geçerken birden fazla olasılık oluşturur. Bu olasılıklardan herhangi bir dalı kabul edilebilir bir state’e ulaşırsa bu NDTM o dili sağlıyor demektir.

1. **NDTM ve DTM Farkları**

* DTM Hesaplama olarak tek bir işlem zincirini takip eder. NDTM ise paralel işlem yapar.
* DTM’ de dallanma yoktur. NDTM ise her bir hücre için farklı dallar açabilir.
* DTM girdiyi adım adım işler ve reddeder. NDTM ise tüm dalları test eder eğer tüm dallar reddediyorsa o dili reddeder.
* DTM daha yavaş olabilir. NDTM Teorik olarak daha hızlıdır.
* DTMler dünyada fiziksel olarak bulunabilir. Fakat NDTMler tamamen teoriktir dünyada bulunmaz.

1. **NDTM ve DTM Eşdeğerliliği**

**Teorem:**

Her Nondeterministic Turing Machine (NDTM) tarafından tanınabilen bir dil, bir Deterministic Turing Machine (DTM) tarafından da tanınabilir. Yani, NDTM’nin tanıyabileceği her dilin DTM tarafından tanınabilmesi mümkündür. Matematiksel olarak:

L(NDTM)=L(DTM)

Bu, NDTM'nin DTM'ye göre daha güçlü olmadığı, ancak farklı bir işlem yöntemiyle aynı hesaplama gücüne sahip olduğunu gösterir.

L={a^nb^n∣n>0} dili aşağıda DTM ve NDTM tarafından tanımlanmıştır

DTM: çizim, diyagram, taslak, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

NDTM:

çizim, diyagram, taslak, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

-------------------------------------------------------------------------------------------------------------------------

**Halting problem**

**Bilgisayar Yetkinlik Teorisinde Durdurma Problemi**

Bilgisayar yetkinlik teorisinde, durdurma problemi, rastgele bir bilgisayar programının ve bir girdinin tanımından yola çıkarak, programın çalışmayı bitirip bitirmeyeceğini ya da sonsuza kadar çalışmaya devam edip etmeyeceğini belirleme problemidir. Durdurma problemi kararsızdır, yani tüm olası program-girdi çiftleri için bu problemi çözen genel bir algoritma yoktur. Bu problem, hesaplanabilirlik tartışmalarında sıkça karşımıza çıkar çünkü bazı fonksiyonların matematiksel olarak tanımlanabilir olduğunu, ancak hesaplanamaz olduğunu gösterir.

**Problemin Matematiksel Tanımı ve Kanıtı**

Problemin resmi ifadesinin önemli bir kısmı, bilgisayar ve programın matematiksel bir tanımını gerektirir; bu genellikle bir Turing makinesi aracılığıyla yapılır. Kanıt, herhangi bir program *f*'nin, programların durup durmayacağını belirleyebileceği varsayımıyla, *f*'nin yanlış bir karar verdiği "patolojik" bir program *g*'nin var olduğunu gösterir. Spesifik olarak, *g*, bir girdi ile çağrıldığında, kendi kaynak kodunu ve girdisini *f*'ye iletip *f*'nin *g* hakkında öngördüğünün tam tersini yapan bir programdır. *g* üzerinde *f*'nin davranışı, kararsızlığı ortaya koyar, çünkü bu durum, hiçbir program *f*'nin durdurma problemini her durumda çözmeyeceği anlamına gelir.

**Arka Plan**

Durma problemi, sabit bir Turing-tam modelinde bilgisayar programlarının özellikleri hakkında bir karar problemidir, yani belirli bir programlama dilinde yazılabilecek ve bir Turing makinesine eşdeğer olacak kadar genel tüm programlar için geçerlidir. Problem, bir program ve bu programa bir girdi verildiğinde, programın bu girdiyi çalıştırdığında nihayetinde durup durmayacağını belirlemektir. Bu soyut çerçevede, programın yürütülmesi için gereken bellek veya zaman miktarı konusunda herhangi bir kaynak sınırlaması yoktur; program durmadan önce keyfi derecede uzun sürebilir ve keyfi miktarda depolama alanı kullanabilir. Soru sadece, verilen programın belirli bir girdi üzerinde bir gün durup durmayacağıdır.

Örneğin, şu sözde kodda yazılan program:

while (true) continue

hiçbir zaman durmaz; sonsuz bir döngüde devam eder. Öte yandan, şu program:

[print "Hello, world!"](https://en.wikipedia.org/wiki/%22Hello,_World!%22_program)

durur.

Bu programların durup durmadığını belirlemek basit olsa da, daha karmaşık programlar sorun yaratır. Bu probleme yönelik bir yaklaşım, programı belirli bir adım sayısı boyunca çalıştırmak ve durup durmadığını kontrol etmek olabilir. Ancak, program çalışmaya devam ettiği sürece, bir gün durup durmayacağı veya sonsuza kadar çalışıp çalışmayacağı bilinemez. Turing, herhangi bir keyfi program ve girdi için programın o girdiyle çalıştırıldığında durup durmayacağını her zaman doğru bir şekilde belirleyen bir algoritmanın mevcut olmadığını kanıtlamıştır. Turing'in kanıtının özeti, bu tür bir algoritmanın çelişkili çıktılar üretmeye zorlanabileceği ve bu nedenle doğru olamayacağıdır.

**Programlama Sonuçları**

Bazı sonsuz döngüler oldukça faydalı olabilir. Örneğin, olay döngüleri (event loops) genellikle sonsuz döngüler olarak kodlanır. Ancak, çoğu alt programın (subroutine) tamamlanması amaçlanır. Özellikle, gerçek zamanlı hesaplamada (hard real-time computing), programcılar yalnızca tamamlanacağı garanti edilen değil, aynı zamanda belirli bir son teslim süresinden önce tamamlanacağı garanti edilen alt programlar yazmaya çalışırlar.

Bazen bu programcılar genel amaçlı (Turing-tam) bir programlama dili kullanır, ancak MISRA C veya SPARK gibi, sonuçta ortaya çıkan alt programların belirlenen süre içinde tamamlanacağını kanıtlamayı kolaylaştıran kısıtlı bir tarzda yazmaya çalışırlar.  
Diğer zamanlarda, bu programcılar en az güç kuralını (rule of least power) uygular—bilerek tam anlamıyla Turing-tam olmayan bir bilgisayar dili kullanırlar. Çoğunlukla, bu diller tüm alt programların tamamlanacağını garanti eden dillerdir, örneğin Coq.

**Durma Problemindeki Zorluklar**

Durma problemindeki zorluk, karar prosedürünün tüm programlar ve girdiler için çalışmasını gerektirmesidir. Belirli bir program, verilen bir girdi üzerinde ya durur ya da durmaz. "Durur" cevabını her zaman veren bir algoritma ile "Durmaz" cevabını her zaman veren bir başka algoritmayı düşünelim. Belirli bir program ve girdi için, bu iki algoritmadan biri doğru cevap verir, ancak hangisinin doğru olduğunu kimse bilmeyebilir. Yine de, bu algoritmaların hiçbiri durma problemini genelde çözemez.

Herhangi bir kaynak kodunu çalıştıran programları (yorumlayıcılar) ele alalım. Bu tür programlar, eğer verilen bir program durursa, bu durumu gösterebilir: Yorumlayıcı, simülasyonunu sonunda durdurur ve bu, orijinal programın durduğunu gösterir. Ancak, bir yorumlayıcı, girdi programı durmazsa kendisi de durmaz; bu nedenle, bu yaklaşım durma problemini çözemez. Çünkü durmayan programlar için "Durmaz" cevabını başarılı bir şekilde veremez.

Durma problemi, teorik olarak, lineer sınırlı otomatlar (LBA'lar) veya sonlu belleğe sahip deterministik makineler için çözülebilirdir. Sonlu belleğe sahip bir makine, sonlu sayıda konfigürasyona sahiptir ve bu nedenle üzerindeki herhangi bir deterministik program nihayetinde ya durmalı ya da önceki bir konfigürasyonu tekrar etmelidir:  
... herhangi bir sonlu durum makinesi, tamamen kendi haline bırakılırsa, en sonunda tamamen periyodik bir tekrar eden bir desene düşecektir. Bu tekrar eden desenin süresi, makinenin iç durumlarının sayısını aşamaz ...

Ancak, iki duruma sahip bir milyon küçük parçadan oluşan bir bilgisayarın en az 21,000,000 olası durumu olur:[5]  
Bu, yaklaşık üç yüz bin sıfırın ardından gelen bir 1’dir ... Böyle bir makine kozmik ışınların frekanslarında çalışsa bile, galaktik evrimin çağları, böyle bir döngüde geçen zamanla kıyaslandığında hiçbir şey gibi kalır:

Bir makine sonlu olabilir ve sonlu otomatlar "bir dizi teorik sınırlamaya sahip olabilir":  
... bu büyüklükler, durum diyagramının yalnızca sonlu olmasına dayanan teoremler ve argümanların büyük bir öneme sahip olmayabileceğini düşündürmelidir.

Sonlu belleğe sahip bir nondeterministik makinenin, hiçbir olasılık dizisinde, bazı olasılık dizilerinde veya tüm olasılık dizilerinde durup durmayacağını otomatik olarak belirlemek de mümkündür. Bu, her olası karar sonrası durumları listeleyerek yapılabilir.

**Biçimselleştirme**

Turing, orijinal kanıtında algoritma kavramını Turing makineleriyle biçimsel hale getirmiştir. Ancak, elde edilen sonuç yalnızca Turing makinelerine özgü değildir; Turing makineleriyle hesaplama gücü açısından eşdeğer olan diğer tüm hesaplama modelleri için de geçerlidir. Bu modeller arasında Markov algoritmaları, Lambda hesaplaması, Post sistemleri, kayıt makineleri ve işaret sistemleri gibi modeller bulunur.

Burada önemli olan, biçimselleştirmenin algoritmaların çalışabileceği bir veri türüne doğrudan bir eşleme yapılmasına izin vermesidir. Örneğin, eğer bir biçimsel sistem algoritmaların diziler üzerinde işlevler tanımlamasına olanak tanıyorsa (örneğin Turing makineleri), bu algoritmaların dizilere bir eşlemesi olmalıdır. Eğer biçimsel sistem algoritmaların doğal sayılar üzerinde işlevler tanımlamasına olanak tanıyorsa (örneğin hesaplanabilir işlevler), bu algoritmaların doğal sayılara bir eşlemesi olmalıdır.

Dizilere yapılan eşleme genellikle en doğrudan olanıdır, ancak n karakterden oluşan bir alfabedeki diziler, bu dizilerin n’lik bir sayı sisteminde sayılar olarak yorumlanmasıyla da sayılara eşlenebilir.

**Set olarak Temsil edilişi**

Karar problemlerinin geleneksel temsili, ilgili özelliğe sahip nesnelerin kümesidir. Durma problemi şu küme ile temsil edilir:

*K* = {(*i*, *x*) | program *i* halts when run on input *x*}

Bu küme, yinelemeli olarak sıralanabilir bir kümedir; yani, içerdiği tüm (i, x) çiftlerini listeleyen hesaplanabilir bir işlev vardır. Ancak, bu kümenin tamamlayanı yinelemeli olarak sıralanabilir değildir.

Durma probleminin birçok eşdeğer formülasyonu vardır; Turing derecesi durma probleminin Turing derecesine eşit olan herhangi bir küme bu tür bir formülasyondur. Bu kümelere örnekler:

* {*i* | program *i* eventually halts when run with input 0}
* {*i* | there is an input *x* such that program *i* eventually halts when run with input *x*}.

**Kanıt Konsepti**

Christopher Strachey, durma probleminin çözülemeyeceğini çelişki yoluyla kanıtlamıştır.Kanıt şu şekilde ilerler:

Halts(*f*) adında bir toplam hesaplanabilir işlev olduğunu ve bu işlevin, alt program ***f*** (girdi olmadan çalıştırıldığında) duruyorsa "true", durmuyorsa "false" döndürdüğünü varsayalım. Şimdi aşağıdaki alt programı düşünelim:

**def** ***g***():

**if** halts(g):

loop\_forever()

*Halts(g), toplam olduğu varsayıldığından, ya "true" ya da "false" döndürmelidir.*

* *Eğer Halts(g) "true" döndürürse, ggg sonsuz döngüye girer ve asla durmaz, bu bir çelişkidir.*
* *Eğer Halts(g) "false" döndürürse, ggg durur çünkü sonsuz döngüye girmez; bu da bir çelişkidir.*

*Sonuç olarak, ggg, Halts işlevinin ggg için söylediğinin tersini yapar, bu nedenle Halts(g), ggg'nin durup durmadığıyla tutarlı bir gerçek değer döndüremez. Bu nedenle, Halts'in toplam hesaplanabilir bir işlev olduğu varsayımı yanlış olmalıdır.*

***Program iii****, sabit bir Turing-tam hesaplama modeli üzerindeki tüm programların bir sıralamasındaki iii. programı ifade eder.*

h(i,x)={1if  program i halts on input x,0otherwise.

Burada program *i*, sabit bir Turing-tam hesaplama modeli üzerindeki tüm programların bir sıralamasındaki *i*. programı ifade eder.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *f*(*i*,*j*) | | *i* | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| *j* | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 | 0 |
|  |  |  |  |  |  |  |  |
|  | *f*(*i*,*i*) | 1 | 0 | 0 | 1 | 1 | 0 |
|  | *g*(*i*) | U | 0 | 0 | U | U | 0 |

Toplam bir hesaplanabilir işlev *f*'nin olası değerleri bir 2D tablo şeklinde düzenlenmiştir. Turuncu hücreler, tablodaki köşegeni (diagonal) temsil eder. Tablonun alt kısmında ***f(i,i)*** ve ***g(i)'nin*** değerleri gösterilmektedir; "U" harfi, *g*'nin belirli bir girdi değeri için tanımlı olmadığını ifade eder.

Kanıt, iki argümanlı toplam bir hesaplanabilir işlevin *h* olması gerektiğini doğrudan reddederek ilerler. Kavramın taslağında olduğu gibi, herhangi bir toplam hesaplanabilir ikili işlev *f* verildiğinde, aşağıdaki kısmi işlev *g* de bir program *e* tarafından hesaplanabilir:

*g*'nin hesaplanabilir olduğu doğrulaması, aşağıdaki yapılar (veya eşdeğerleri) kullanılarak gerçekleştirilir:

* **Hesaplanabilir alt programlar** (*f*'yi hesaplayan program, *e* programında bir alt program olarak kullanılır),
* **Değerlerin çoğaltılması** (***e, g*** için verilen iii girdisinden *f* için ***i,ii, ii,i*** girdilerini hesaplar),
* **Koşullu dallanma** (*e, f(i,i)* için hesapladığı değere bağlı olarak iki sonuçtan birini seçer),
* **Tanımlı bir sonuç üretmeme** (örneğin, sonsuz döngüye girerek),
* **Bir değeri döndürme** (örneğin, 0).

Aşağıdaki sözde kod, *e*'nin *g*'yi hesaplamak için nasıl çalıştığını basit bir şekilde gösterir:

**procedure** e(i):

**if** f(i, i) == 0 **then**

return 0

**else**

loop forever

Çünkü *g* kısmi hesaplanabilir bir işlevdir, hesaplama modelinin Turing-tam olduğu varsayımıyla *g*'yi hesaplayan bir program *e* olmalıdır.  
Bu program, durma işlevi *h*'nin tanımlandığı tüm programlardan biridir. Kanıtın bir sonraki adımı, *h(e,e)*'nin *f(e,e)* ile aynı değere sahip olamayacağını göstermektir.

*g*'nin tanımından, aşağıdaki iki durumdan tam olarak birinin geçerli olması gerektiği sonucu çıkar:

* *f*(*e*,*e*) = 0 and so *g*(*e*) = 0. In this case program *e* halts on input *e*, so *h*(*e*,*e*) = 1.
* *f*(*e*,*e*) ≠ 0 and so *g*(*e*) is undefined. In this case program *e* does not halt on input *e*, so *h*(*e*,*e*) = 0.

Her iki durumda da *f*, *h* ile aynı işlev olamaz. Çünkü *f*, iki argümanlı herhangi bir toplam hesaplanabilir işlev olarak keyfi şekilde seçilmişti, bu tür tüm işlevler *h*'den farklı olmalıdır.

Bu kanıt, Cantor'un köşegen argümanına benzer.  
Doğal sayıların her biri için bir sütun ve bir satır içeren iki boyutlu bir tablo hayal edilebilir, yukarıdaki tabloda belirtildiği gibi. *f(i,j)*'nin değeri, sütun *i* ve satır *j* kesişiminde yer alır. *f*'nin toplam hesaplanabilir bir işlev olduğu varsayıldığından, tablonun herhangi bir elemanı *f* kullanılarak hesaplanabilir. *g* işlevinin oluşturulması, bu tablonun ana köşegeni kullanılarak görselleştirilebilir. Eğer tablonun *(i,i)* konumunda bir 0 varsa, *g(i)=0*. Aksi takdirde, *g(i)* tanımsızdır.

Çelişki, tablodaki bir sütunun *g*'ye karşılık geldiği bir sütun *e* bulunmasından kaynaklanır. Eğer *f*, durma işlevi *h* olarak varsayılırsa:

* Eğer *g(e)*tanımlıysa (*g(e)=0* bu durumda), *g(e)* durur ve dolayısıyla *f(e,e)=1* . Ancak *g(e)=0*, yalnızca *f(e,e)=0* olduğunda tanımlanır, bu da *f(e,e)=1* ile çelişir.
* Benzer şekilde, eğer *g(e)* tanımlı değilse, durma işlevi *f(e,e)=0*  olur, bu da *g*'nin yapısı gereği *g(e)=0'ı* doğurur. Bu, *g(e)* 'nin tanımlı olmadığı varsayımıyla çelişir.

Her iki durumda da çelişki ortaya çıkar. Bu nedenle, herhangi bir keyfi hesaplanabilir işlev *f*, durma işlevi *h* olamaz.

**König'in lemması**

Kőnig lemması veya Kőnig'in sonsuzluk lemması, 1927'de yayınlayan Macar matematikçi [Dénes Kőnig'e](https://en.wikipedia.org/wiki/D%C3%A9nes_K%C5%91nig" \o "Denes König) ait bir [graf teorisi](https://en.wikipedia.org/wiki/Graph_theory) teoremidir .Sonsuz bir grafın sonsuz uzunlukta bir yola sahip olması için yeterli bir koşul sağlar. Bu teoremin hesaplanabilirlik yönleri , özellikle [hesaplanabilirlik teorisinde ,](https://en.wikipedia.org/wiki/Recursion_theory)[matematiksel mantıktaki](https://en.wikipedia.org/wiki/Mathematical_logic) araştırmacılar tarafından kapsamlı bir şekilde araştırılmıştır . Bu teoremin ayrıca [yapıcı matematik](https://en.wikipedia.org/wiki/Constructive_mathematics) ve [kanıt teorisinde](https://en.wikipedia.org/wiki/Proof_theory) önemli rolleri vardır .

**Lemmanın ifadesi**

İzin vermek  [bağlı](https://en.wikipedia.org/wiki/Connected_graph) , [yerel olarak sonlu](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms#finite) , [sonsuz](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms#finite) bir grafik olsun. Bu, her iki köşenin sonlu bir yolla bağlanabileceği, her köşenin yalnızca sonlu sayıda başka köşeye bitişik olduğu ve grafiğin sonsuz sayıda köşeye sahip olduğu anlamına gelir. Sonra  [bir ışın](https://en.wikipedia.org/wiki/Ray_(graph_theory)) içerir : tek bir köşeden başlayıp sonsuz sayıda köşeden devam eden [basit bir yol (tekrarlanan köşesi olmayan bir yol).](https://en.wikipedia.org/wiki/Path_(graph_theory))

Lemmanın kullanışlı bir özel durumu, her sonsuz [ağacın](https://en.wikipedia.org/wiki/Tree_(graph_theory)) ya sonsuz [derecede](https://en.wikipedia.org/wiki/Degree_(graph_theory)) bir tepe noktası ya da sonsuz basit bir yol içermesidir. Yerel olarak sonluysa, lemmanın koşullarını karşılar ve bir ışına sahiptir ve yerel olarak sonlu değilse, sonsuz derecede bir tepe noktasına sahiptir.

**Yapı**

Bir ışının bir grafikte inşası  lemmanın koşullarını karşılayan, adım adım gerçekleştirilebilir, her adımda sonsuz sayıda köşeye (hepsi birbiriyle aynı yol boyunca olmak zorunda değil) ulaşmak için uzatılabilen sonlu bir yol korunur. Bu işleme başlamak için herhangi bir tek köşeden başlayın . Bu tepe noktası, sıfır uzunlukta, tek bir tepe noktasından ve kenarlardan oluşan bir yol olarak düşünülebilir. Lemmanın varsayımlarına göre, sonsuz sayıdaki tepe noktasının her biri  basit bir yoldan ulaşılabilir .

Sonra, mevcut yol bir köşe noktasında bittiği sürece , geçerli yolu uzatan basit yollarla ulaşılabilen sonsuz sayıda köşeyi düşünün ve bu köşelerin her biri için geçerli yolu uzatan basit bir yol oluşturun. Sonsuz sayıda bu uzatılmış yol vardır ve her biri komşularından birine, ama [yalnızca sonlu sayıda komşusu vardır. Bu nedenle, güvercin deliği ilkesinin](https://en.wikipedia.org/wiki/Pigeonhole_principle) bir biçimine göre  bu komşulardan en az birinin bu genişletilmiş yolların sonsuz sayıdaki bir sonraki adımı olarak kullanıldığı sonucu çıkar. böyle bir komşu ol ve mevcut yolu bir kenar kadar uzat, ile Bu uzantı, geçerli yolu genişleten basit yollarla sonsuz sayıda köşeye ulaşılabileceği özelliğini korur.

Yolu uzatmak için bu işlemi tekrarlamak, her biri dizideki önceki yolu bir kenar daha uzatan sonsuz bir sonlu basit yol dizisi üretir. Tüm bu yolların birleşimi, lemma tarafından varlığı vaat edilen ışındır.

**Hesaplanabilirlik yönleri**

Kőnig lemmasının hesaplanabilirlik yönleri kapsamlı bir şekilde araştırılmıştır. Bu amaçla Kőnig lemmasını herhangi bir sonsuz sonlu dallanma alt ağacının biçiminde belirtmek uygundur. ω<ω sonsuz bir yolu vardır. Buradaω w doğal sayılar kümesini ( [sıralı sayı](https://en.wikipedia.org/wiki/Ordinal_number) olarak düşünülen ) belirtir ve düğümleri doğal sayıların sonlu dizileri olan ağaç, burada bir düğümün ebeveyni bir diziden son elemanın çıkarılmasıyla elde edilir. Her sonlu dizi, kısmi bir fonksiyonla tanımlanabilir w kendisine ve her sonsuz yol toplam bir fonksiyonla tanımlanabilir. Bu, hesaplanabilirlik teorisinin tekniklerini kullanarak bir analize olanak tanır.

Bir alt ağaç  her bir dizinin yalnızca sonlu sayıda doğrudan uzantısının olduğu (yani, ağacın bir grafik olarak bakıldığında sonlu bir derecesi olduğu) sonlu dallanma olarak adlandırılır . Her sonsuz alt ağacın  sonsuz bir yola sahiptir, ancak König lemması herhangi bir sonlu dallanan sonsuz alt ağacın böyle bir yola sahip olması gerektiğini gösterir.

Herhangi bir alt ağaç için T ile ilgili notasyonGenişletilmiş(T)  düğüm kümesini belirtirT T içinden sonsuz bir yol geçen. HattaT T küme hesaplanabilir miGenişletilmiş(T)  hesaplanabilir olmayabilir. Bir alt ağaç her ne zaman TT ile ilgili ω<ω sonsuz bir yola sahipse, yol hesaplanabilirGenişletilmiş(T) , adım adım, açgözlülükle bir halef seçerek Genişletilmiş(T) her adımda.  bu açgözlü sürecin tıkanıp kalmamasını sağlar.

Sonlu olmayan dallanmalı hesaplanabilir alt ağaçlar mevcuttur ω<ω [aritmetik](https://en.wikipedia.org/wiki/Arithmetical_hierarchy) yolu olmayan ve aslında [hiperaritmetik](https://en.wikipedia.org/wiki/Analytical_hierarchy" \o "Analitik hiyerarşi) yolu olmayan.  Ancak, her hesaplanabilir alt ağacıω<ω  bir yol ile [Kleene'nin O'sundan](https://en.wikipedia.org/wiki/Kleene%27s_O" \o "Kleene'nin O'su) hesaplanabilir bir yola sahip olmak gerekir , kanonikΠ11  tam set. Bunun nedeni setin Genişletilmiş(T)her zaman (bu gösterimin anlamı için [analitik hiyerarşiye](https://en.wikipedia.org/wiki/Analytical_hierarchy) bakın )T T hesaplanabilir.

Hesaplanabilir şekilde sınırlandırılmış ağaçlar için daha ayrıntılı bir analiz yürütülmüştür. Bir alt ağaç ω<ω hesaplanabilir bir fonksiyon varsa hesaplanabilir sınırlı veya yinelemeli sınırlı olarak adlandırılırF itibarenω  ileω  böylece ağaçtaki her dizi ve her doğal sayı içinN n,N n dizinin en fazla elemanıF(N) f(n) . BöyleceF f ağacın ne kadar "geniş" olduğuna dair bir sınır verir. Aşağıdaki [temel teoremler,](https://en.wikipedia.org/wiki/Basis_theorem_(computability)) sonsuz, hesaplanabilir sınırlı, hesaplanabilir alt ağaçlara uygulanır.ω<ω

* Bu tür ağaçların herhangi birinin hesaplanabilir bir yolu vardır0′ 0’ [Durma problemini](https://en.wikipedia.org/wiki/Halting_problem) çözebilen kanonik Turing tam kümesi .
* Bu tür herhangi bir ağacın yolu düşüktür [. Bu,](https://en.wikipedia.org/wiki/Low_(computability))[düşük baz teoremi](https://en.wikipedia.org/wiki/Low_basis_theorem) olarak bilinir .
* Bu tür herhangi bir ağacın hiperimmun olmayan bir yolu vardır . Bu, yoldan hesaplanabilen herhangi bir fonksiyonun hesaplanabilir bir fonksiyon tarafından domine edildiği anlamına gelir.
* Hesaplanamayan herhangi bir alt küme içinX X ile ilgiliω  ağacın hesaplanmayan bir yolu var X X.

Her sonsuz ikili ağacın sonsuz bir dalı olduğunu belirten Kőnig lemmasının zayıf bir biçimi, [ikinci dereceden aritmetiğin](https://en.wikipedia.org/wiki/Second-order_arithmetic) WKL 0 alt sistemini tanımlamak için kullanılır . Bu alt sistemin [ters matematikte](https://en.wikipedia.org/wiki/Reverse_mathematics) önemli bir rolü vardır . Burada ikili ağaç, ağaçtaki her dizinin her teriminin 0 veya 1 olduğu bir ağaçtır, yani ağaç sabit fonksiyon 2 aracılığıyla hesaplanabilir şekilde sınırlandırılmıştır. Kőnig lemmasının tam biçimi WKL 0'da kanıtlanamaz ancak daha güçlü ACA0’a eşdeğerdir .

**Yapıcı Matematik ve Kompaktlık ile İlişkisi**

Yukarıda verilen kanıt genellikle yapıcı olarak kabul edilmez, çünkü her adımda sonsuz sayıda başka köşeye ulaşılabilen bitişik bir köşenin varlığını kanıtlamak için çelişki yoluyla bir kanıt kullanır ve seçim aksiyomunun zayıf bir biçimine güvenir. Lemmanın hesaplamalı yönleri hakkındaki gerçekler, yapıcı matematiğin ana okulları tarafından yapıcı olarak kabul edilebilecek hiçbir kanıtın verilemeyeceğini düşündürmektedir.

LEJ Brouwer'ın (1927) fan teoremi, klasik bir bakış açısından, Kőnig lemmasının bir biçiminin karşıtıdır. Bir alt küme herhangi bir fonksiyondan bir çubuk denir, set ω üzerinde {0,1} öğelerinin bir başlangıç segmentine sahiptir. Her dizi ya bardaysa ya da barda değilse bir bar ayrılabilirdir (bu varsayım, teoremin genellikle hariç tutulan orta kuralının varsayılmadığı durumlarda dikkate alınması nedeniyle gereklidir). Bir bar, bir sayı varsa tekdüzedir ***N***, böylece herhangi bir fonksiyon ω ile {0,1} çubukta uzunluğu en fazla ***N*** olmayan bir başlangıç segmenti vardır. Brouwer'in fan teoremi, ayrılabilir her çubuğun düzgün olduğunu söyler.

Bu, çubuğun kompakt topolojik uzayın açık bir örtüsü olarak düşünülmesiyle klasik bir ortamda kanıtlanabilir .Çubuktaki her dizi bu uzayın temel açık kümesini temsil eder ve bu temel açık kümeler varsayıma göre uzayı kaplar. Kompaktlık nedeniyle, bu örtünün sonlu bir alt örtüsü vardır. Fan teoreminin ***N***'si, temel açık kümesi sonlu alt örtüde olan en uzun dizinin uzunluğu olarak alınabilir. Bu topolojik kanıt, klasik matematikte Kőnig lemmasının şu biçiminin geçerli olduğunu göstermek için kullanılabilir: herhangi bir doğal sayı ***k*** için, ağacın herhangi bir sonsuz alt ağacı sonsuz bir yolu vardır.

**Seçim aksiyomu ile ilişki**

Kőnig'in lemması bir seçim ilkesi olarak düşünülebilir; yukarıdaki ilk kanıt, lemma ile [bağımlı seçim aksiyomu](https://en.wikipedia.org/wiki/Axiom_of_dependent_choice) arasındaki ilişkiyi göstermektedir . Tümevarımın her adımında, belirli bir özelliğe sahip bir tepe noktası seçilmelidir. En az bir uygun tepe noktasının var olduğu kanıtlanmış olsa da, birden fazla uygun tepe noktası varsa kanonik seçim olmayabilir. Aslında, bağımlı seçim aksiyomunun tam gücüne ihtiyaç yoktur; aşağıda açıklandığı gibi, [sayılabilir seçim aksiyomu](https://en.wikipedia.org/wiki/Axiom_of_countable_choice) yeterlidir.

Eğer grafik sayılabilirse, köşeler iyi sıralanmıştır ve en küçük uygun köşe kanonik olarak seçilebilir. Bu durumda, Kőnig'in lemması [aritmetiksel kavrayışa](https://en.wikipedia.org/wiki/Arithmetical_comprehension) sahip ikinci dereceden aritmetikte ve a fortiori, [ZF küme teorisinde](https://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory) (seçim olmaksızın) kanıtlanabilir.

**Sonuç ve Değerlendirme**

Bu çalışmada Nondeterministic Turing Makinelerinin (NDTM) dil karar verme yetenekleri ve deterministik Turing makineleri (DTM) ile arasındaki eşdeğerlik detaylı bir şekilde incelenmiştir. Örnek uygulamalardan elde edilen veriler, NDTM'lerin paralel hesaplama yapabilme yeteneği sayesinde bazı durumlarda pratik avantajlar sağladığını göstermektedir. Ancak teorik olarak DTM'lerle eşdeğer oldukları kanıtlanmıştır. Sonuç olarak, bu çalışma NDTM'nin teorik gücünü ve DTM ile arasındaki ilişkiyi anlamada önemli bir katkı sağlamıştır. Gelecek çalışmalarda bu makinelerin farklı alanlardaki uygulamaları ve sınırlamaları daha detaylı incelenebilir.

**Referanslar**

* [*Brouwer, L. E. J.*](https://en.wikipedia.org/wiki/Luitzen_Egbertus_Jan_Brouwer)*(1927), On the Domains of Definition of Functions*. published in *van Heijenoort, Jean, ed. (1967), From Frege to Gödel*
* *Franchella, Miriam (1997), "On the origins of Dénes König's infinity lemma", Archive for History of Exact Sciences,****51****(51(1)3:2-3): 3–27,*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/BF00376449*](https://doi.org/10.1007%2FBF00376449)*,*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*117198918*](https://api.semanticscholar.org/CorpusID:117198918)
* *Howard, Paul; [Rubin, Jean](https://en.wikipedia.org/wiki/Jean_E._Rubin" \o "Jean E. Rubin) (1998), Consequences of the Axiom of Choice, Mathematical Surveys and Monographs, vol. 59, Providence, RI: American Mathematical Society*
* [*Kőnig, D.*](https://en.wikipedia.org/wiki/D%C3%A9nes_K%C5%91nig)*(1927),*[*"Über eine Schlussweise aus dem Endlichen ins Unendliche"*](https://archive.today/20141223141608/http:/acta.fyx.hu/acta/showCustomerArticle.action?id=5131&dataObjectType=article&returnAction=showCustomerVolume&sessionDataSetId=2b29ea26fa2c9ba&style=)*, Acta Sci. Math. (Szeged) (in German),****3****(2–3): 121–130, archived from [the original](http://acta.fyx.hu/acta/showCustomerArticle.action?id=5131&dataObjectType=article&returnAction=showCustomerVolume&sessionDataSetId=2b29ea26fa2c9ba&style=) on 2014-12-23, retrieved 2014-12-23*
* [*Lévy, Azriel*](https://en.wikipedia.org/wiki/Azriel_L%C3%A9vy)*(1979), Basic Set Theory, Springer,*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*3-540-08417-7*](https://en.wikipedia.org/wiki/Special:BookSources/3-540-08417-7)*,*[*MR*](https://en.wikipedia.org/wiki/MR_(identifier))[*0533962*](https://mathscinet.ams.org/mathscinet-getitem?mr=0533962); reprint, Dover, 2002, [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-486-42079-5](https://en.wikipedia.org/wiki/Special:BookSources/0-486-42079-5).
* [*Rogers, Hartley Jr.*](https://en.wikipedia.org/wiki/Hartley_Rogers,_Jr.)*(1967), Theory of Recursive Functions and Effective Computability, McGraw-Hill,*[*MR*](https://en.wikipedia.org/wiki/MR_(identifier))[*0224462*](https://mathscinet.ams.org/mathscinet-getitem?mr=0224462)
* [*Truss, J.*](https://en.wikipedia.org/wiki/John_Truss)*(1976), "Some cases of König's lemma", in Marek, V. Wiktor; Srebrny, Marian; Zarach, Andrzej (eds.), Set Theory and Hierarchy Theory a Memorial Tribute to Andrzej Mostowski, Lecture Notes in Mathematics, vol. 537, Springer, pp. 273–284,*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/BFb0096907*](https://doi.org/10.1007%2FBFb0096907)*,*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-3-540-07856-2*](https://en.wikipedia.org/wiki/Special:BookSources/978-3-540-07856-2)*,*[*MR*](https://en.wikipedia.org/wiki/MR_(identifier))[*0429557*](https://mathscinet.ams.org/mathscinet-getitem?mr=0429557)