9. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS'15)

Büyüyen Kısmi Yol Kısıtlarıyla Konkolik Test

Yavuz Köroğlu
yavuz.koroglu@boun.edu.tr

Alper Şen
alper.sen@boun.edu.tr

Boğaziçi Üniversitesi
Bilgisayar Mühendisliği Bölümü

September 21, 2015

İçindekiler

Birim Test

Tanım

Bir yazılımın diline göre bir metodunun, bir fonksiyonunun veya bir objesinin icrasını diğer birimlerden bağımsız olarak test etmeye **birim test** denir.

Java ve C

- Java'da yazilan JUnit testleri birer birim testtir.
- C için de CUnit [5] ortamı mevcuttur.

Motivasyon

Günümüzdeki Durum

- Projelerin birim testleri eğer yazılıyorsa genelde **elle** yazılmaktadır.
- Microsoft geliştiricilerinin **79%** u birim test yazımıyla uğraşmaktadır [7].
- Yazılan testlerin **kapsayıcılığı** genelde ölçülmemektedir.

Birim Testleri Otomatik Yaratmak,

- Güvenirliği artıracak,
- İşgücü ihtiyacını azaltacaktır.

Otomatik Testin Geçmişi

İlk Sembolik İcra

- SELECT - 1975, Robert S. Boyer, Bernard Elspas, Karl N. Levitt, Computer Science Group Stanford Research Institute [1]
- EFFIGY - 1976, James C. King, IBM Thomas J. Watson Research Center [4]

İlk Somut İcra

- 1975, Mantıksal devreler üzerinde rastgele test [11]
- 1984, Yazılım üzerinde rastgele test [3]

Sembolik İcra

Yöntem

- Program kodunun statik olarak yorumlanabilir sembolik bir dile çevrilmesi(Ör: LLVM).
- Bu dil üzerinde programın çalıştırılması.
- İcra sırasında toplanan koşulları sağlayan değerler somut icrada da aynı çalışma yoluna götürecektir.
- Program icrasının sembolik bilgisi üç parçadan oluşur:
  1. Önkoşullar
  2. Arakoşullar(Yol Kısıtı)
  3. Sonkoşullar

Sembolik Kısıtların Toplanması

Sembolik değişkenler x_0 ve y_0 olarak el ile belirlenir. Sembolik olarak belirlenmeyen bütün değişkenlere bir değer atanmalıdır.

```
1  int myUnit(int x, int y)
2  {
3      x++;
4      if (x == y)
5          return 1;
6      else
7          return 0;
8  }
```

Önkoşullar:

$x_0 \in \mathbb{Z}, y_0 \in \mathbb{Z}$

Arakoşullar:

$x_1 = x_0 + 1$

$x_1 = y_0$

Sonkoşullar:

$myUnit(x_0, y_0) = 1$

Sembolik Kısıtların Toplanması

Fonksiyonun tanım kümesinden değerlerle çağrılması önkoşuldur.

```
1  int myUnit(int x, int y)
2  {
3      x++;
4      if (x == y)
5          return 1;
6      else
7          return 0;
8  }
```

Önkoşullar:

$x_0 \in \mathbb{Z}, y_0 \in \mathbb{Z}$

Arakoşullar:

$x_1 = x_0 + 1$

$x_1 = y_0$

Sonkoşullar:

$myUnit(x_0, y_0) = 1$

Sembolik Kısıtların Toplanması

Her yeni değer ataması için yeni bir sembolik değişken tanımlanır.

```
1  int myUnit(int x, int y)
2  {
3      x++;
4      if (x == y)
5          return 1;
6      else
7          return 0;
8  }
```

Önkoşullar:

$x_0 \in \mathbb{Z}, y_0 \in \mathbb{Z}$

Arakoşullar:

$x_1 = x_0 + 1$

$x_1 = y_0$

Sonkoşullar:

$myUnit(x_0, y_0) = 1$

Sembolik Kısıtların Toplanması

Dallanmalar birden çok sembolik yol oluşturur. Burada dallanma koşulunun sağlandığı yol ele alınmıştır.

```
1  int myUnit(int x, int y)
2  {
3      x++;
4      if (x == y)
5          return 1;
6      else
7          return 0;
8  }
```

Önkoşullar:

$x_0 \in \mathbb{Z}, y_0 \in \mathbb{Z}$

Arakoşullar:

$x_1 = x_0 + 1$

$x_1 = y_0$

Sonkoşullar:

$myUnit(x_0, y_0) = 1$

Sembolik Kısıtların Toplanması

Sonkoşul: Fonksiyonun döndürmesi gereken değer üzerindeki bütün kısıtlardır.

```
1  int myUnit(int x, int y)
2  {
3      x++;
4      if (x == y)
5          return 1;
6      else
7          return 0;
8  }
```

Önkoşullar:

$x_0 \in \mathbb{Z}, y_0 \in \mathbb{Z}$

Arakoşullar:

$x_1 = x_0 + 1$

$x_1 = y_0$

Sonkoşullar:

$myUnit(x_0, y_0) = 1$

Örnek Kod (CUTE Örneği)

1.Adım: Sembolik değişkenlerin el ile belirlenmesi.

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

Değişkenler: p_0, v_0, n_0, x_0

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

2.Adım: Sembolik icra ile kısıtların toplanması

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_1 = \{x_0 > 0,$
Satır No: 11

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon$

Satır No: 12

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0$

Satır No: 13

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 = p_0$

Satır No: 14

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 = p_0\}$

Satır No: 15

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_2 = \{x_0 > 0,$
Satır No: 11

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon$
Satır No: 12

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0$

Satır No: 13

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 \neq p_0$

Satır No: 14

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 \neq p_0\}$

Satır No: 16

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_3 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 \neq v_0\}$

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Örnek Kod (CUTE Örneği)

$\pi_4 = \{x_0 > 0, p_0 = \epsilon\}$

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

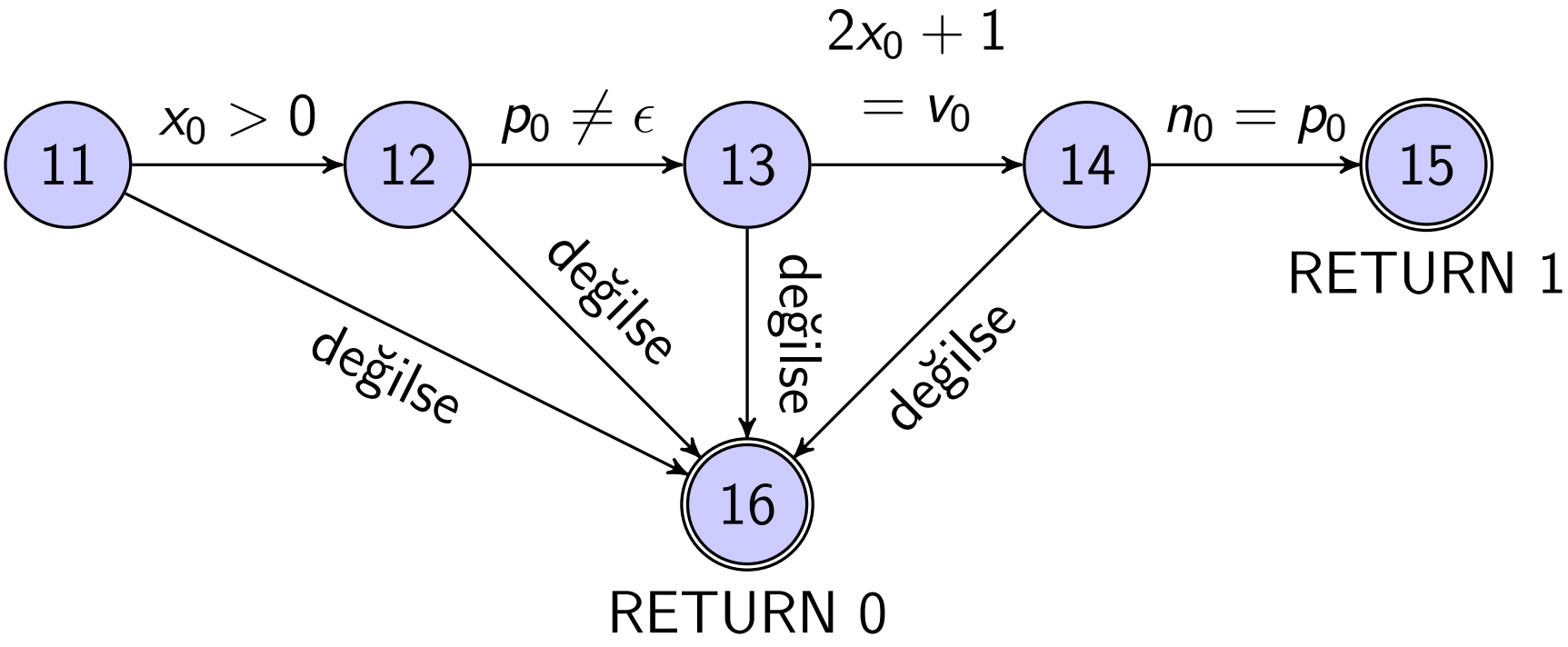
Örnek Kod (CUTE Örneği)

$\pi_5 = \{x_0 \leq 0\}$

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

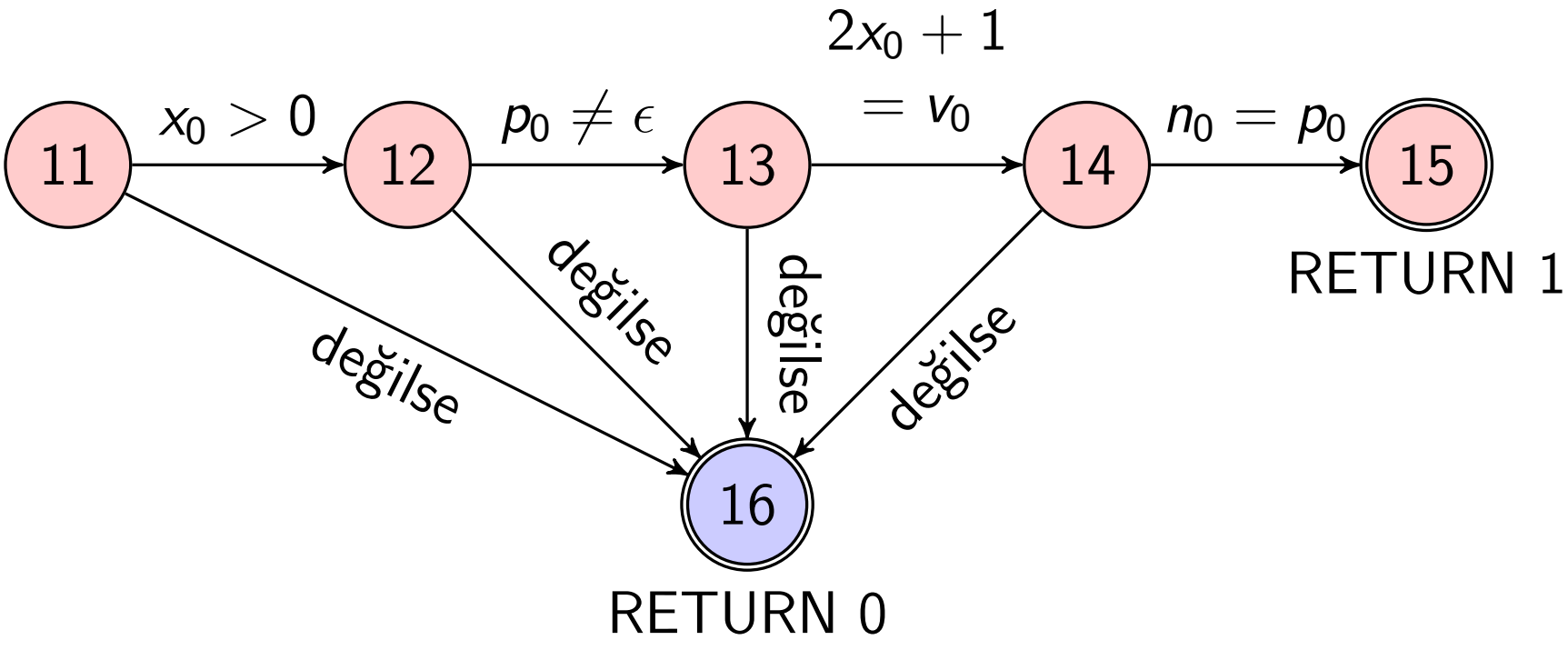
Sembolik İcra Örneği

3.Adım: Kısıt çözücü kullanarak gerekli testleri üretmek



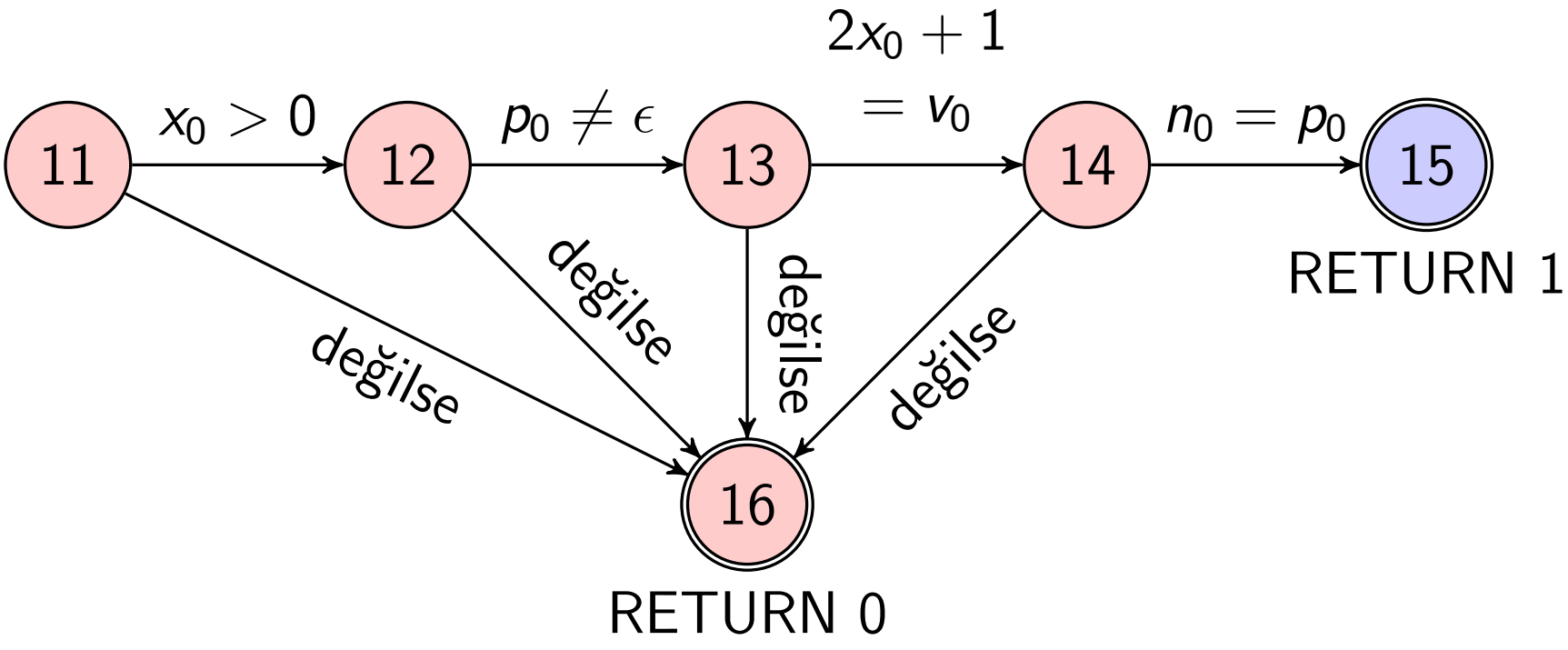
Sembolik İcra Örneği

$$\text{ÇÖZ}(\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 = p_0\})$$
$$= [x_0 = 1, p_0 = 1, v_0 = 3, n_0 = 1]$$



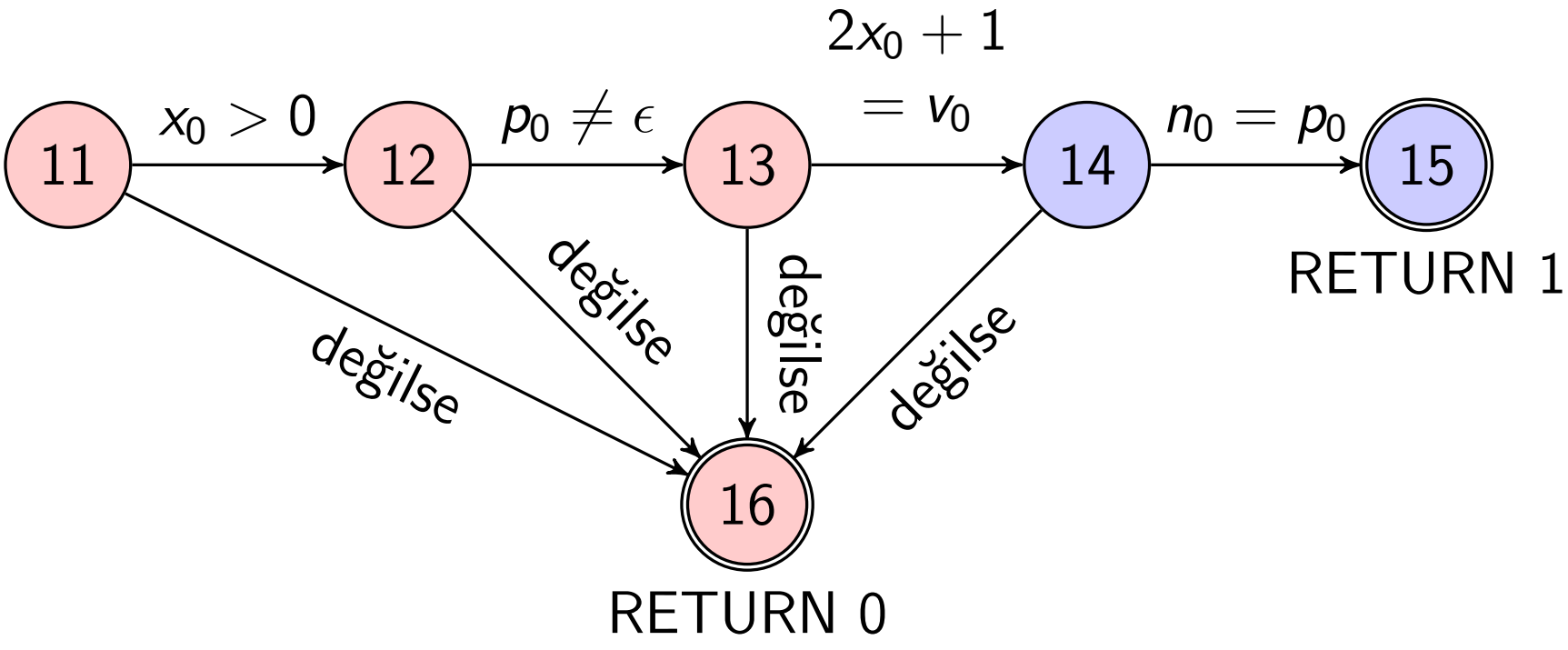
Sembolik İcra Örneği

$$\text{ÇÖZ}(\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 \neq p_0\})$$
$$= [x_0 = 1, p_0 = 1, v_0 = 3, n_0 = 2]$$



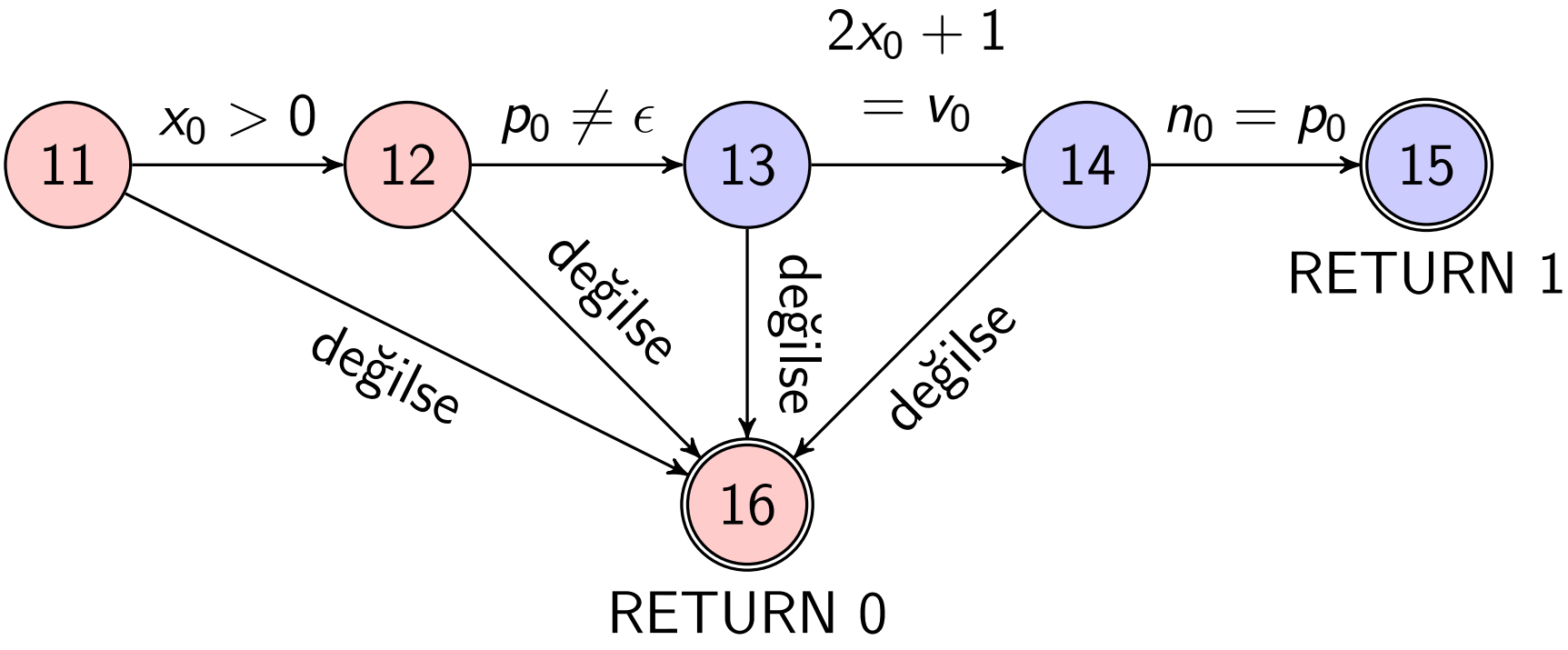
Sembolik İcra Örneği

ÇÖZ($\pi_3 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 \neq v_0\}$)
$= [x_0 = 1, p_0 = 1, v_0 = 4, n_0 = 2]$



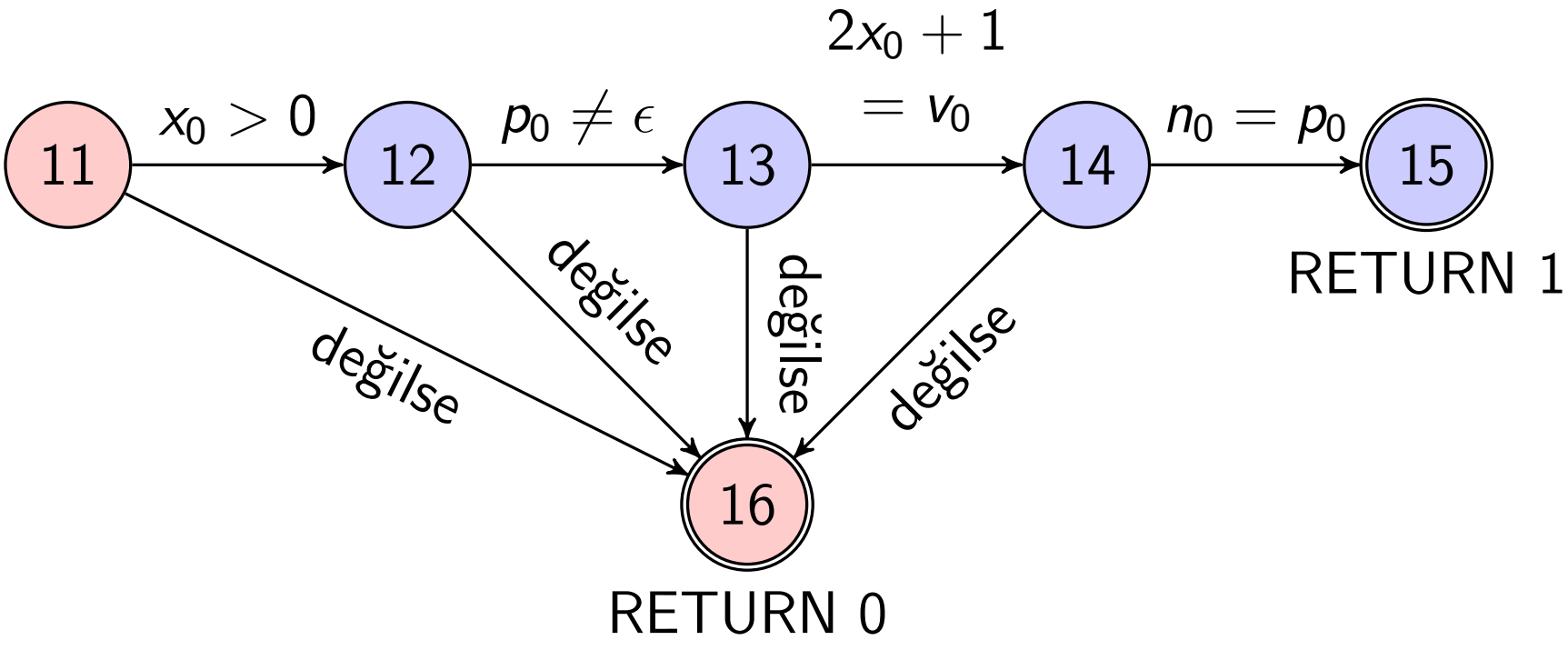
Sembolik İcra Örneği

ÇÖZ($\pi_2 = \{x_0 > 0, p_0 = \epsilon\}$)
$= [x_0 = 1, p_0 = \epsilon, v_0 = 4, n_0 = 2]$



Sembolik İcra Örneği

ÇÖZ($\pi_1 = \{x_0 \leq 0\}$)
$= [x_0 = 1, p_0 = \epsilon, v_0 = 4, n_0 = 2]$



Sembolik İcra Örneği

Sonuçlar

- Kısıt çözücü: 5 kere çağırıldı.
- Kısıt uzunluğu: Ortalama 2.8.
- İterasyon sayısı: 5.

Sorular

- Sembolik değişkenlerin belirlenmesi.
- Kısıtların toplanması.
- Kısıtların çözülmesi.

Sembolik İcra

Sembolik İcrada Geliştirilmeye Açık Yönler

- Ölçeklenebilir hale getirilmesi ve pratikte kullanılması
- Nondeterministik bir icranın test edilmesi
- Kapsayıcılığın artırılması

Konkolik Test

- Somut(concrete) ve sembolik(symbolic) kelimelerinin birleşimi
- Sembolik bir dil üzerinde çalıştırmak yerine somut icralar çalışırken sembolik verilerin dinamik olarak toplanmasi.
- Sembolik verilerin yeni somut icralar icin kullanilmasi.
- Yol sapması ve yol patlaması sorunlarını hafifletirken **kısıt çözücü** yükünü hafifletmemektedir.

Konkolik Testin Geçmişi

- CUTE - 2005, C icin [9]
- PEX - 2008, NET Framework icin [10]
- Janala2(CATG) - 2013, Java icin [6]
- Jalangi - 2013, JavaScript icin [8]

Konkolik İcra Örneği

1.Adım: Sembolik değişkenlerin el ile belirlenmesi.

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

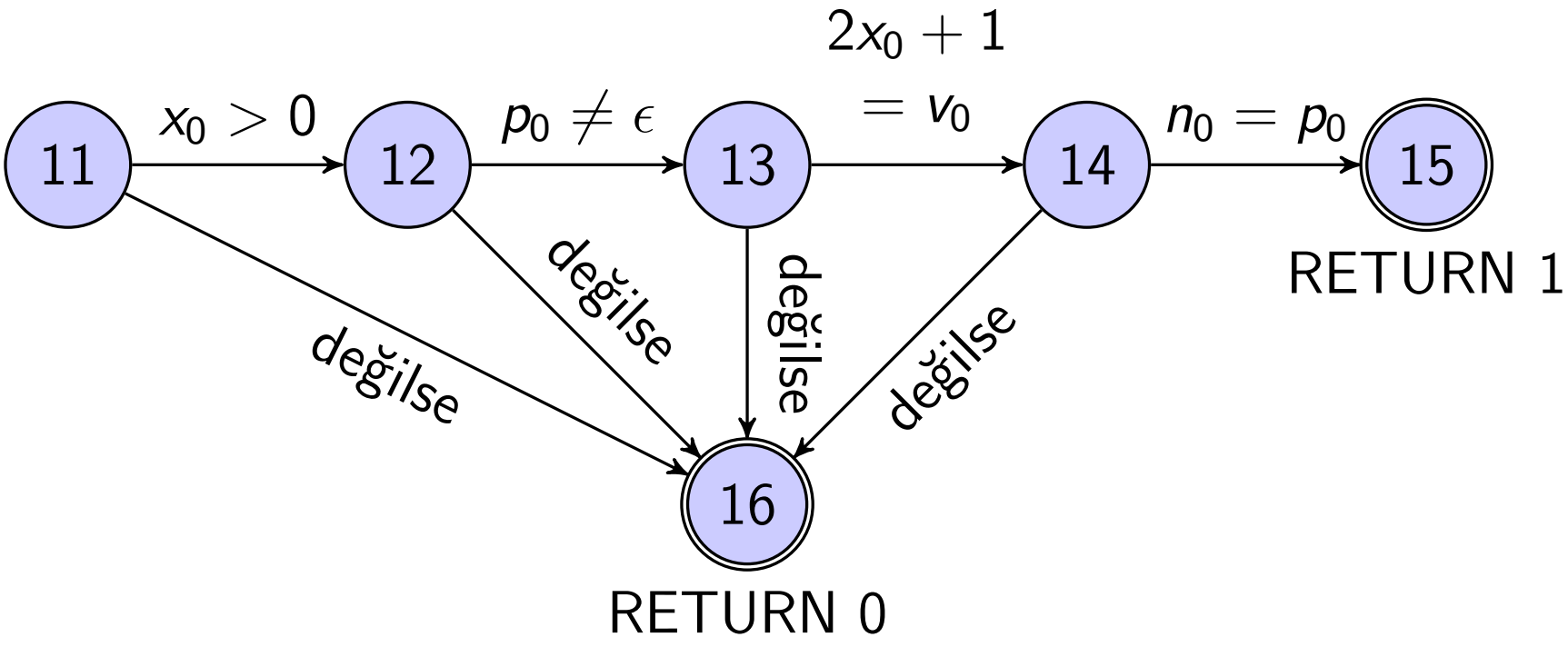
Konkolik İcra Örneği

Değişkenler: p_0, v_0, n_0, x_0

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

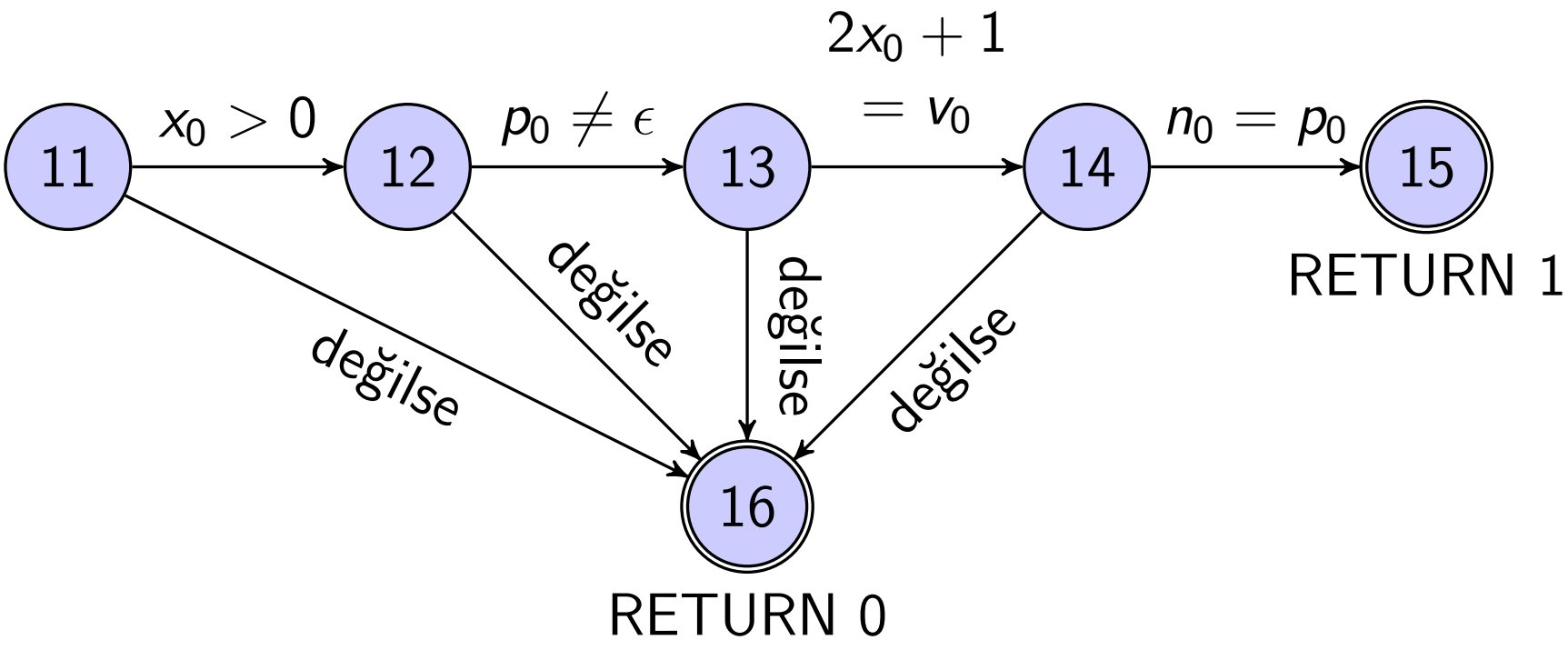
Konkolik İcra Örneği

2.Adım: Sembolik değişkenlere rastgele değerler ata.



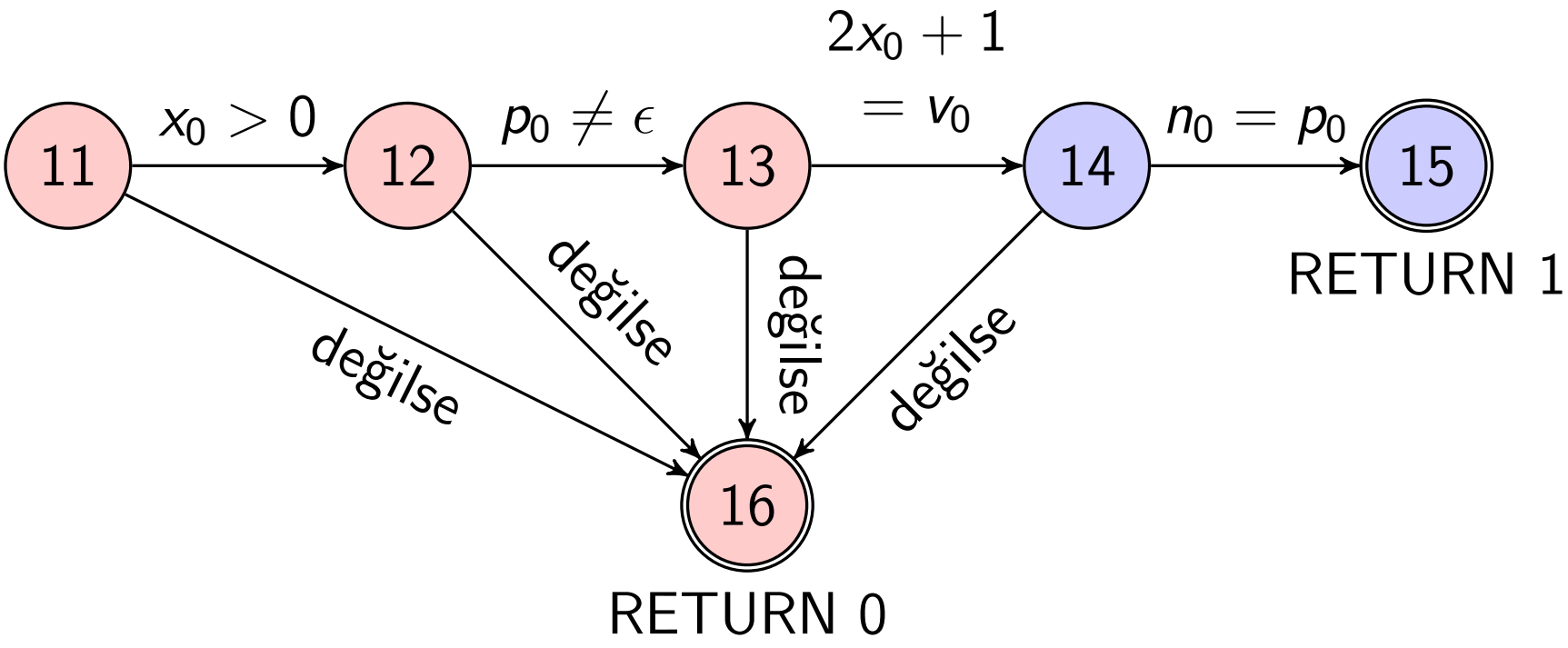
Konkolik İcra Örneği

$[x_0 = 7, p_0 = 9, v_0 = 12, n_0 = 8]$



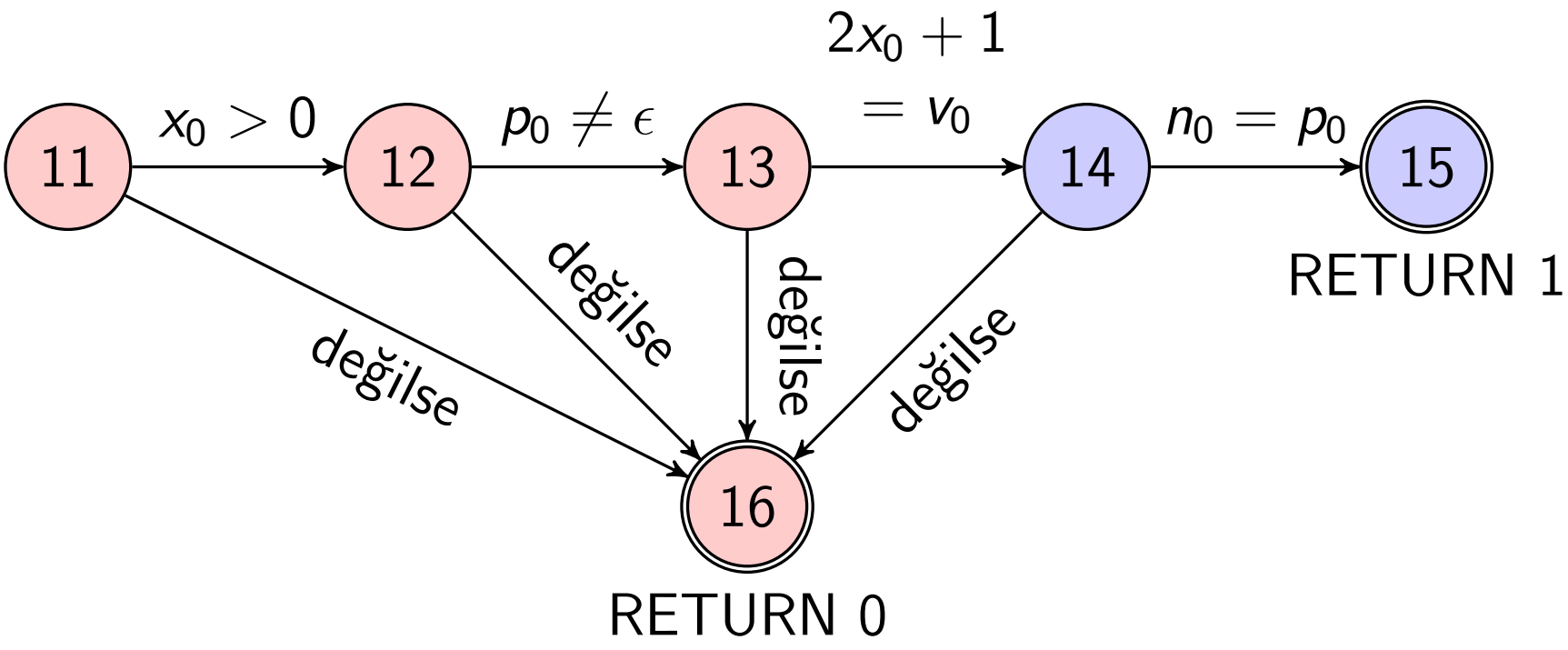
Konkolik İcra Örneği

Programı çalıştır ve sembolik kısıtları topla



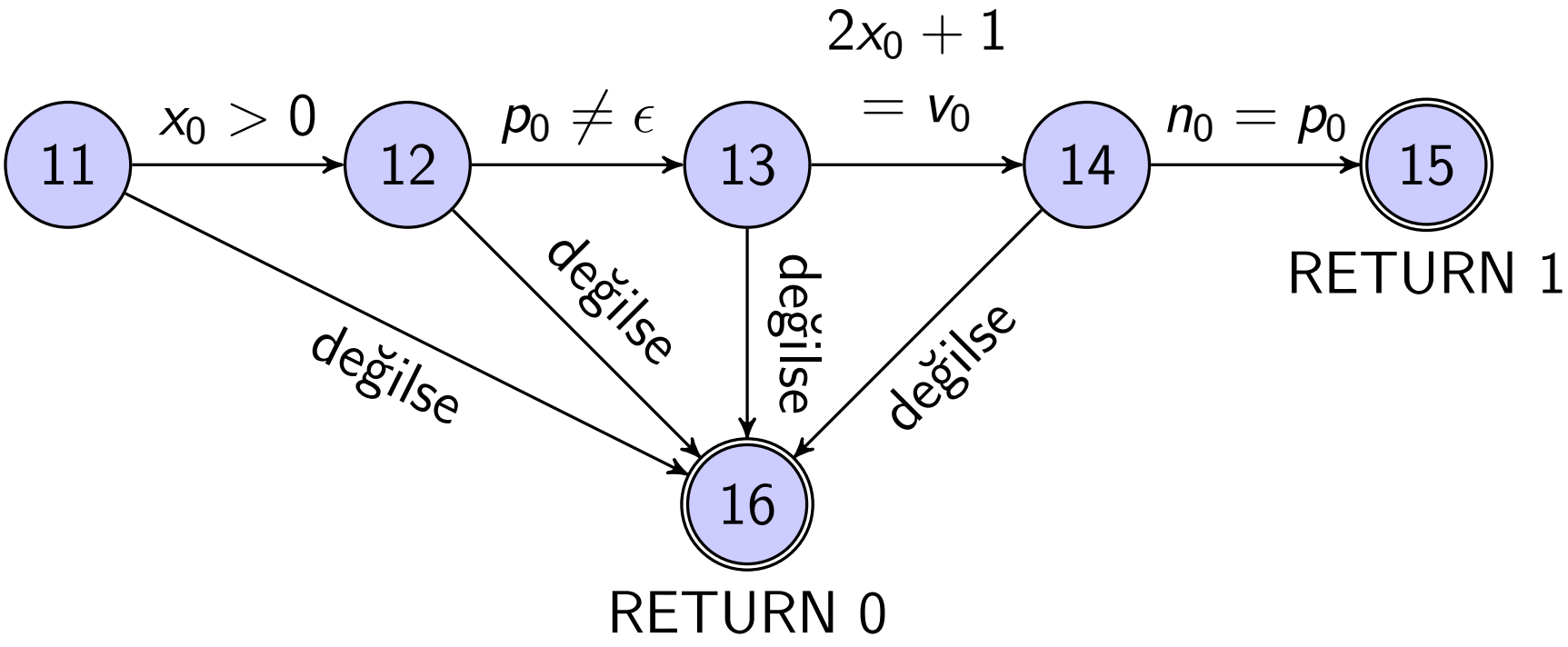
Konkolik İcra Örneği

$\pi_0 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 \neq v_0\}$



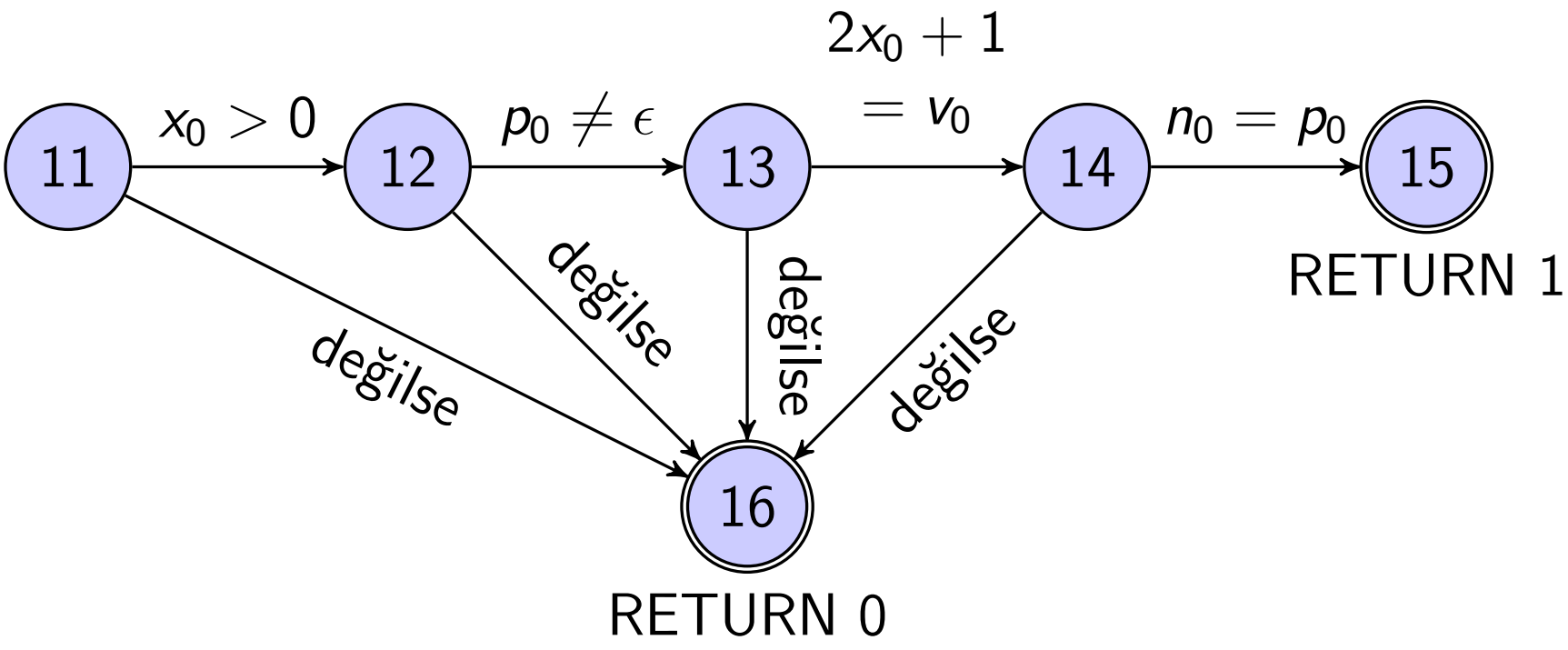
Konkolik İcra Örneği

π_0 kısıtının son koşulunu tersine çevirerek yeni kısıt elde et



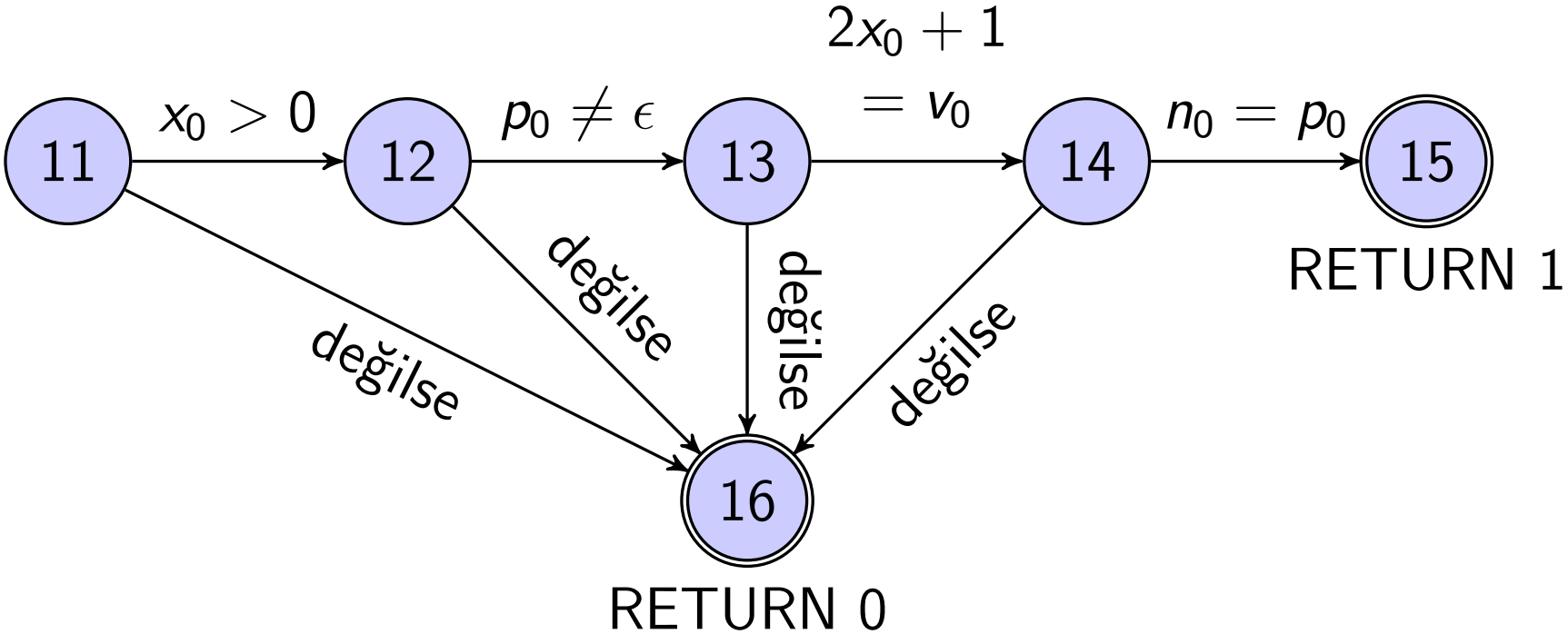
Konkolik İcra Örneği

$$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0\}$$



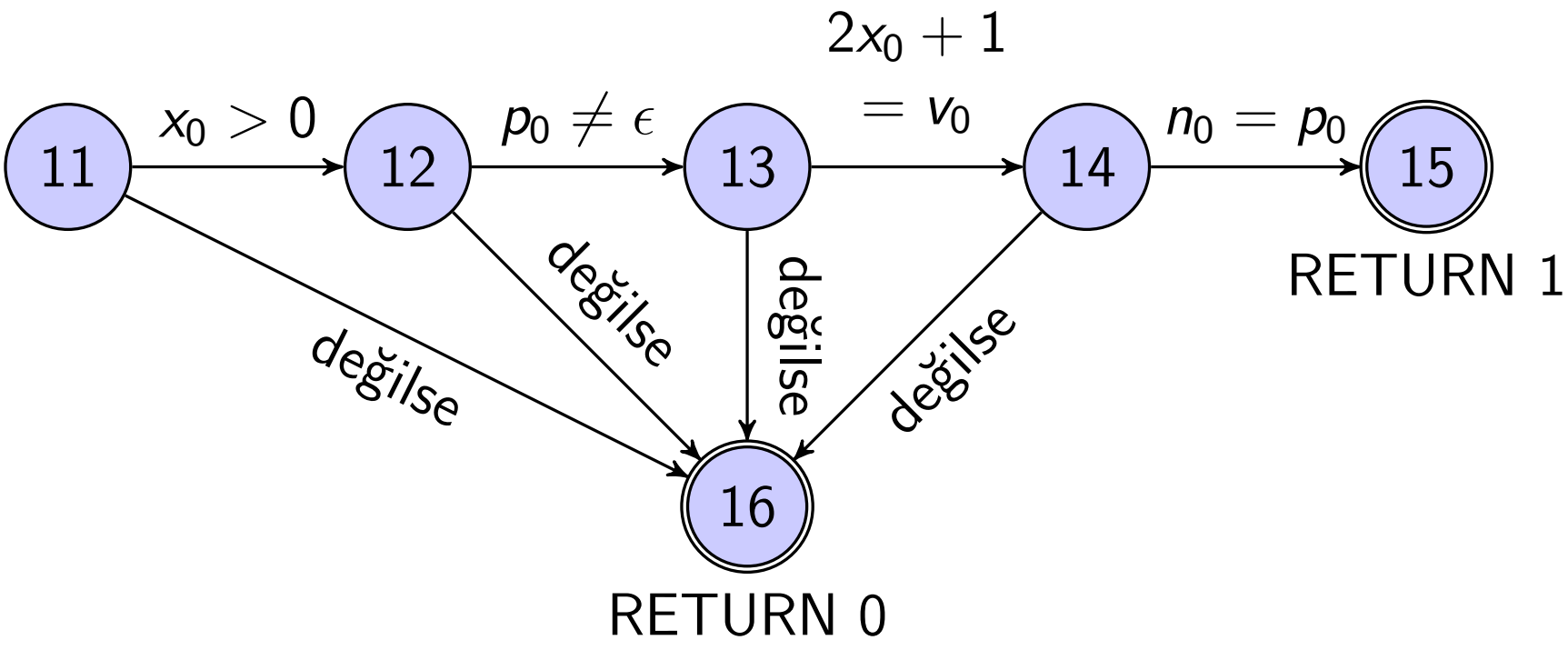
Konkolik İcra Örneği

ÇÖZ($\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0\}$)



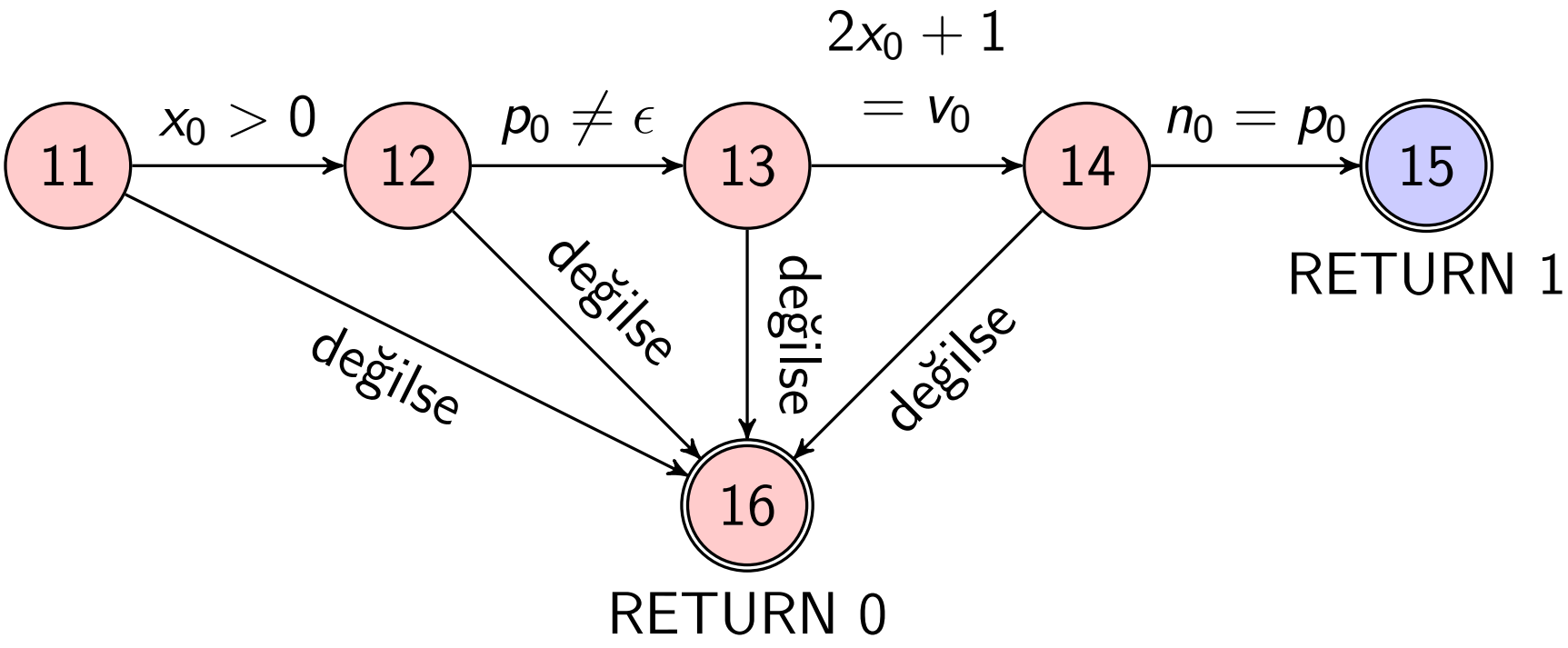
Konkolik İcra Örneği

$= [x_0 = 7, p_0 = \epsilon, v_0 = 15, n_0 = 8]$



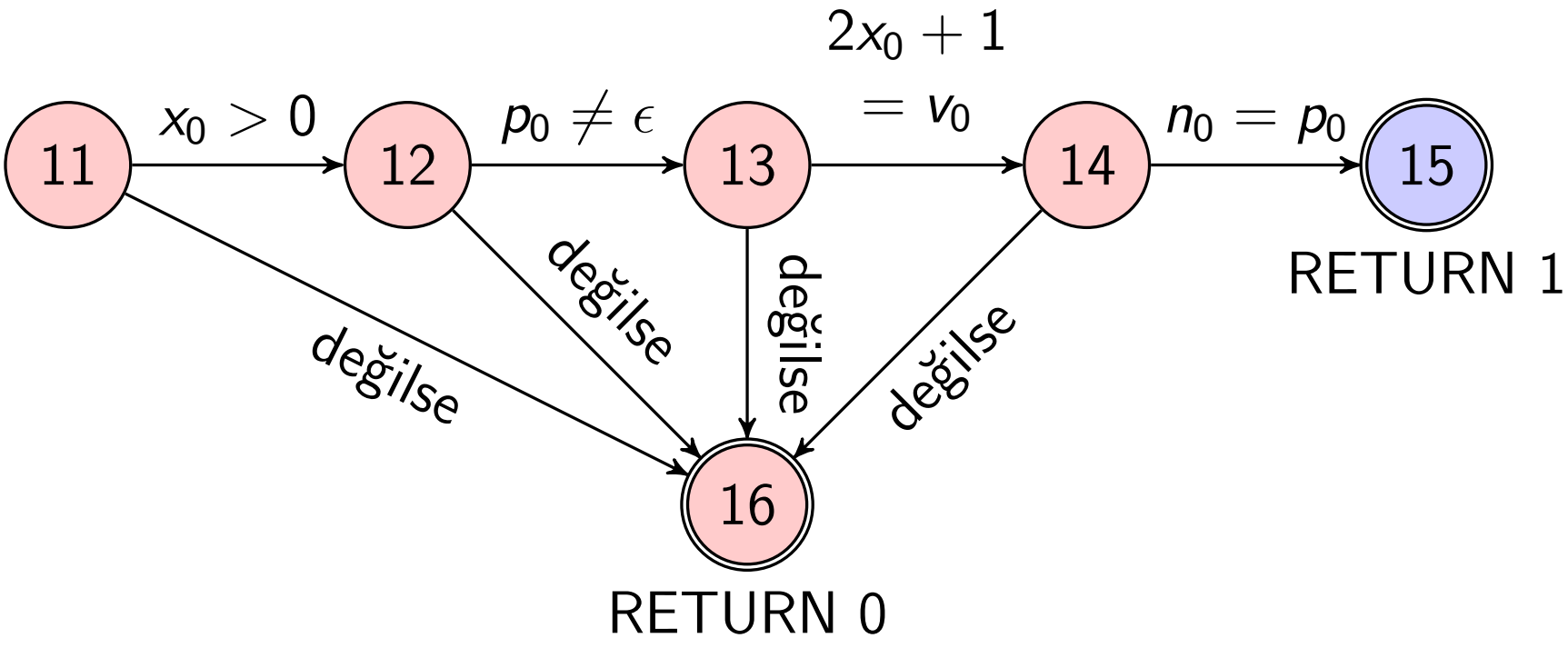
Konkolik İcra Örneği

Programı çalıştır ve sembolik kısıtları topla



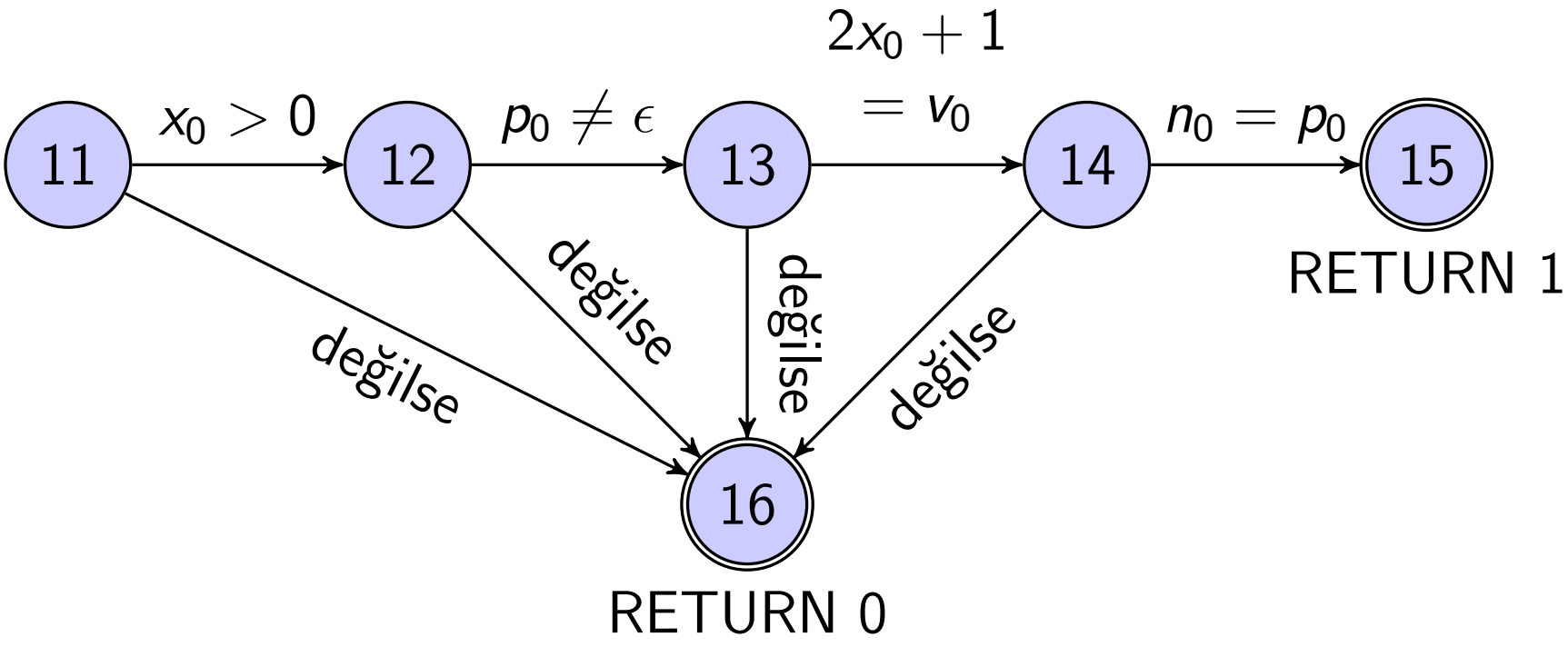
Konkolik İcra Örneği

$$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 \neq n_0\}$$



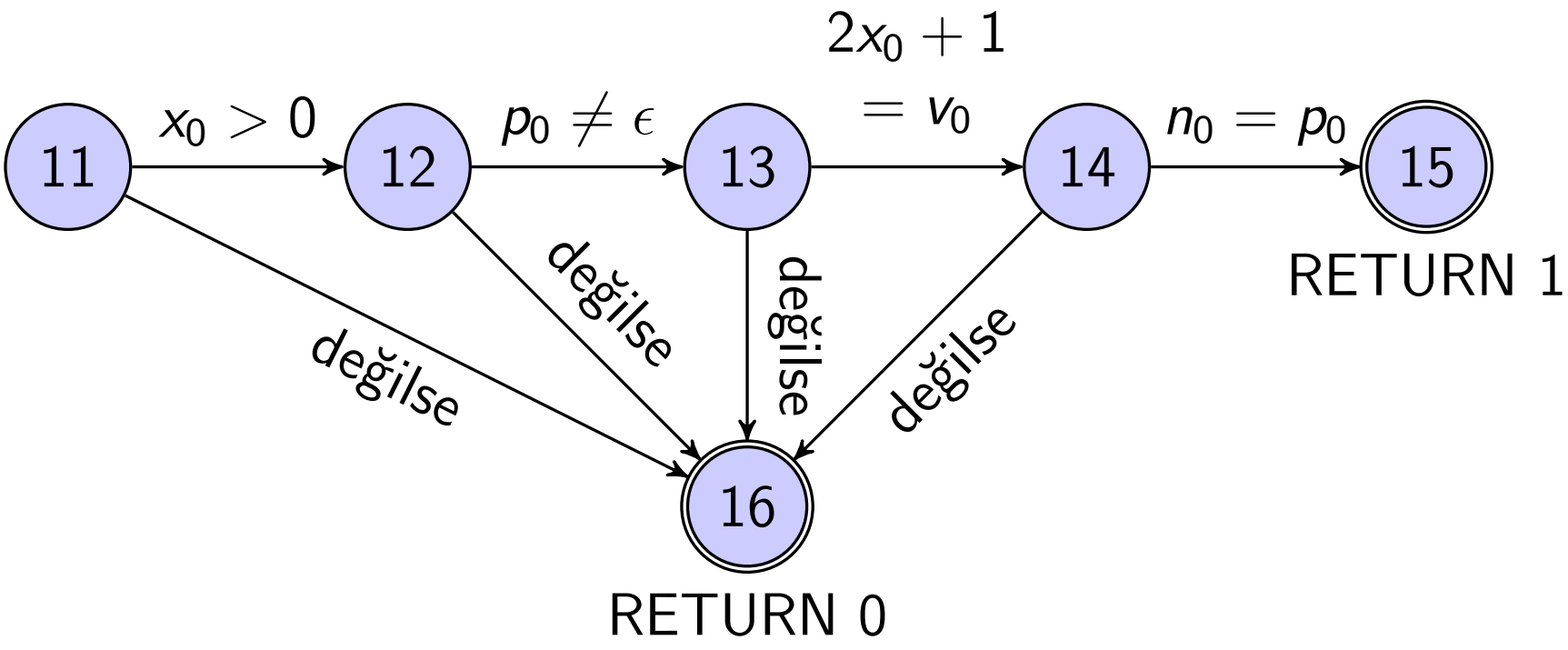
Konkolik İcra Örneği

π_2 kısıtının son koşulunu tersine çevirerek yeni kısıt elde et.



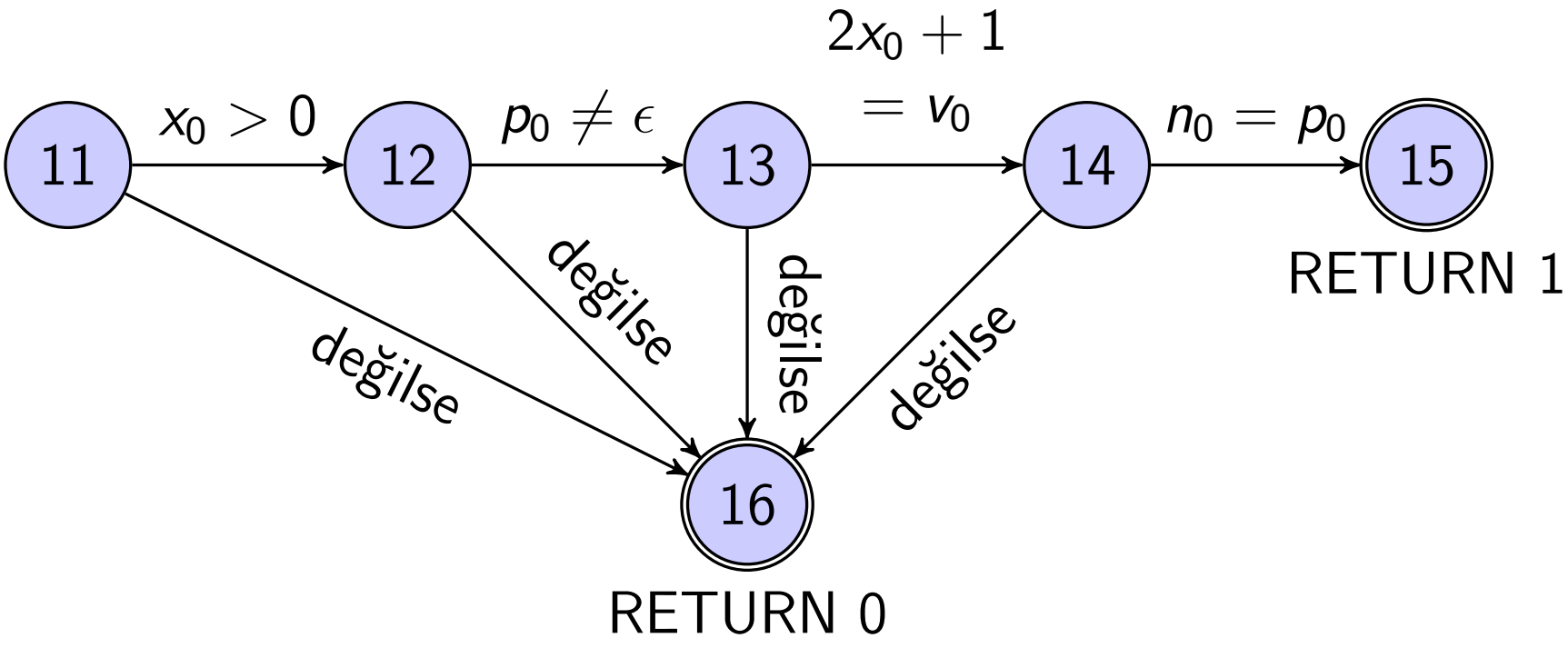
Konkolik İcra Örneği

$$\pi_3 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 = n_0\}$$



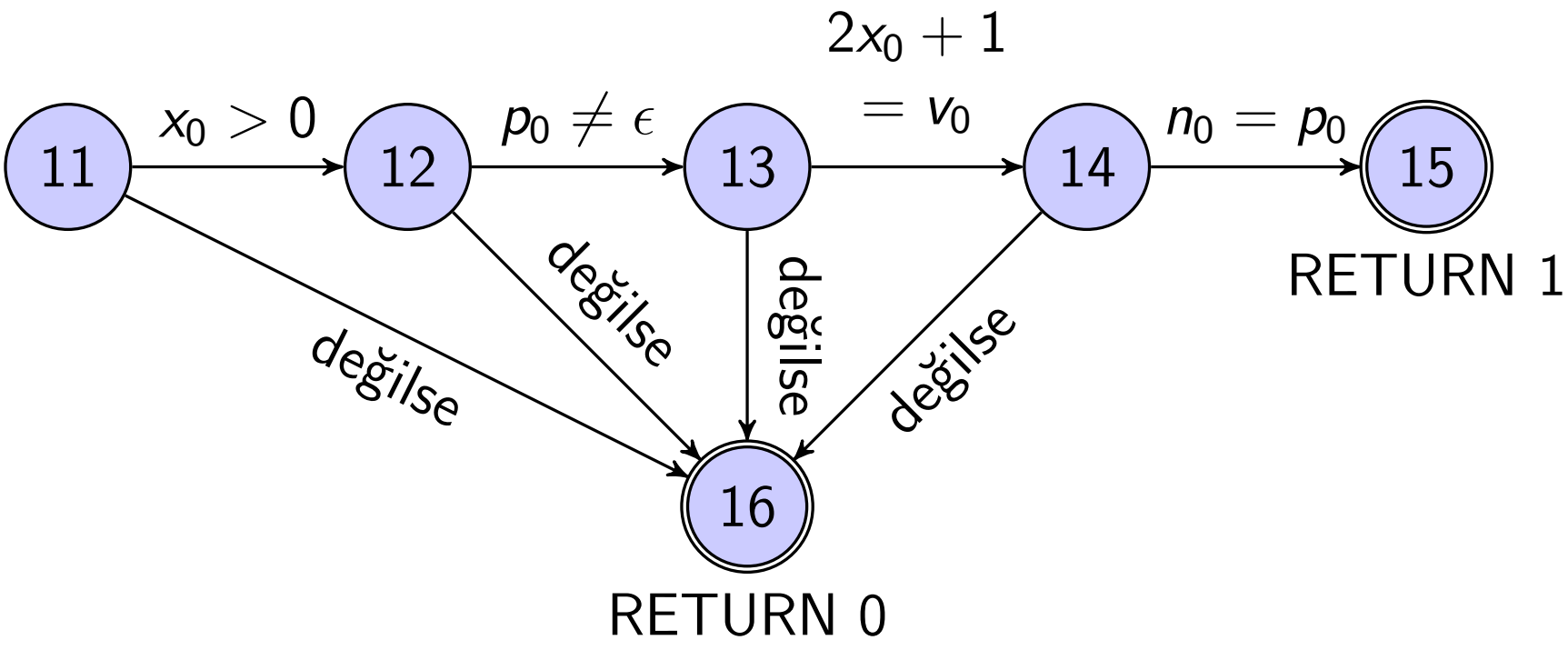
Konkolik İcra Örneği

$$\text{ÇÖZ}(\pi_3 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 = n_0\})$$

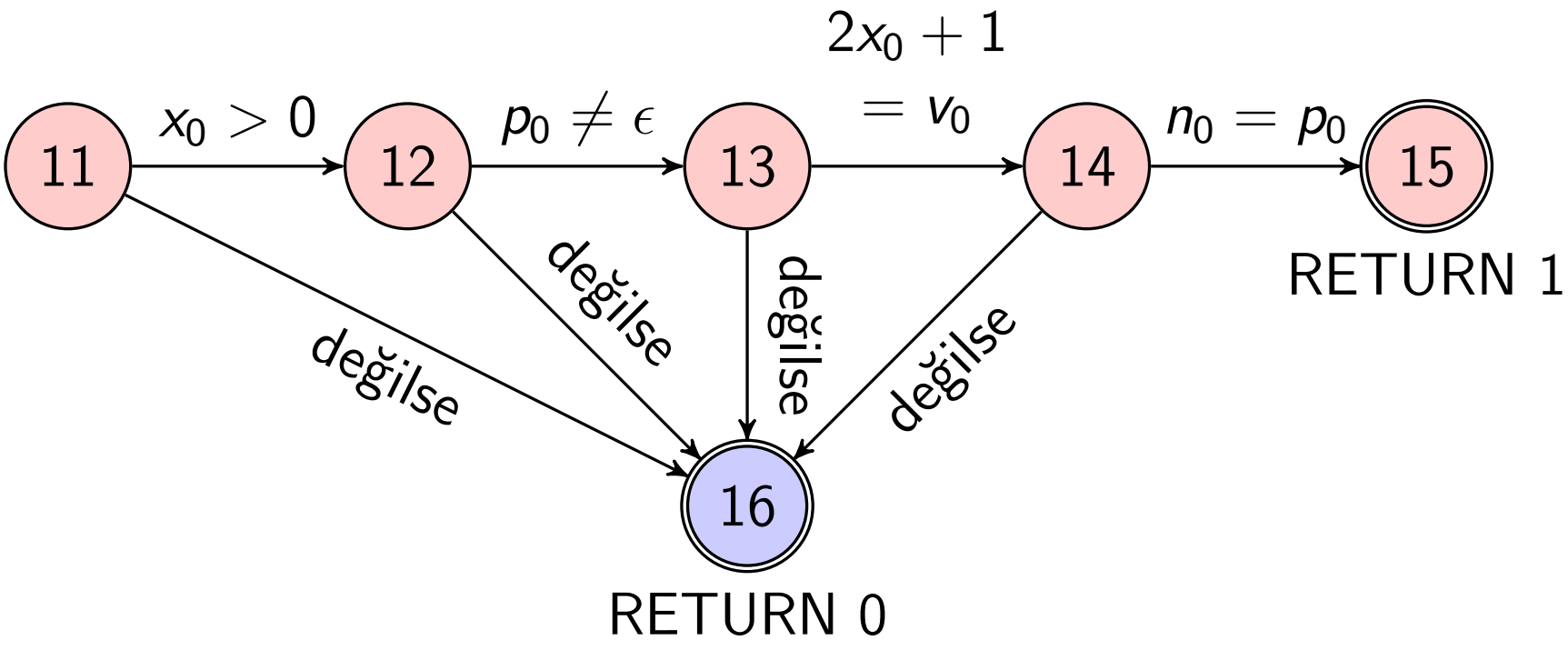


Konkolik İcra Örneği

$= [x_0 = 7, p_0 = 9, v_0 = 15, n_0 = 9]$

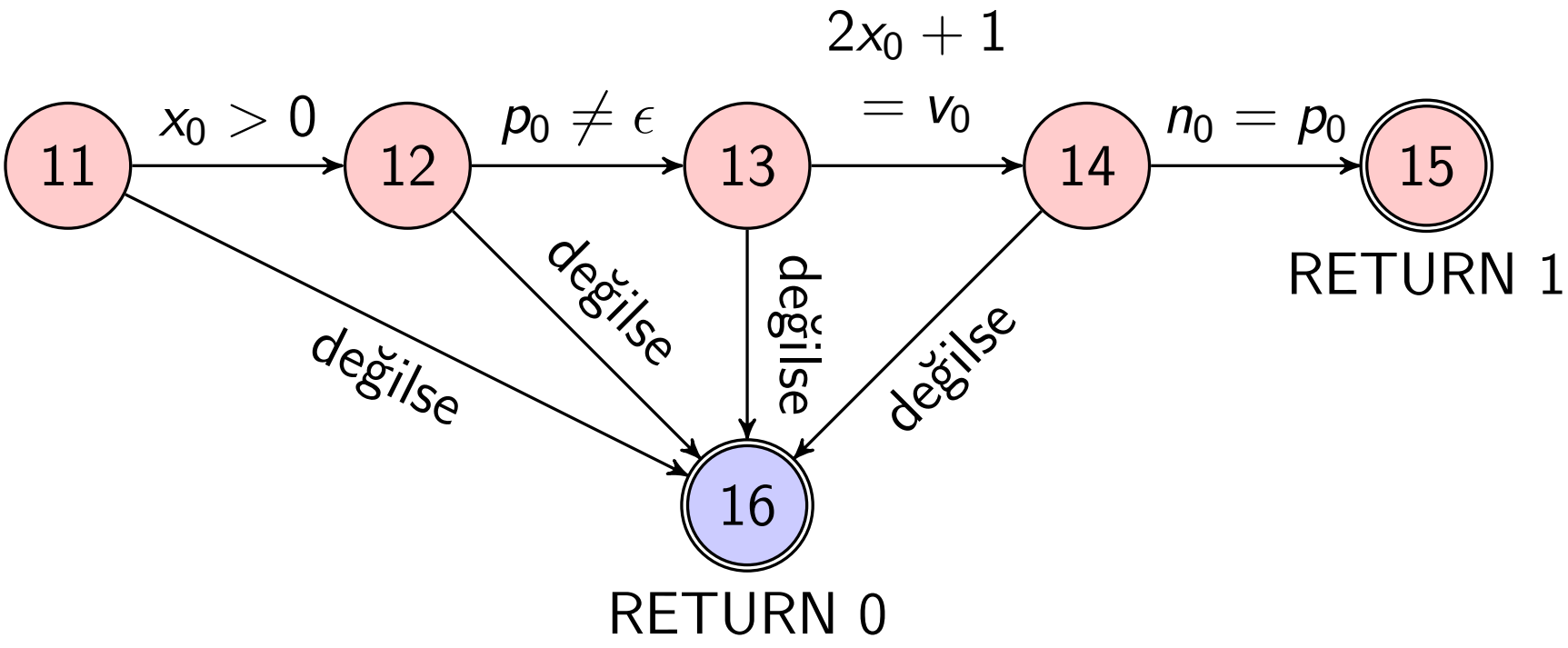


Konkolik İcra Örneği



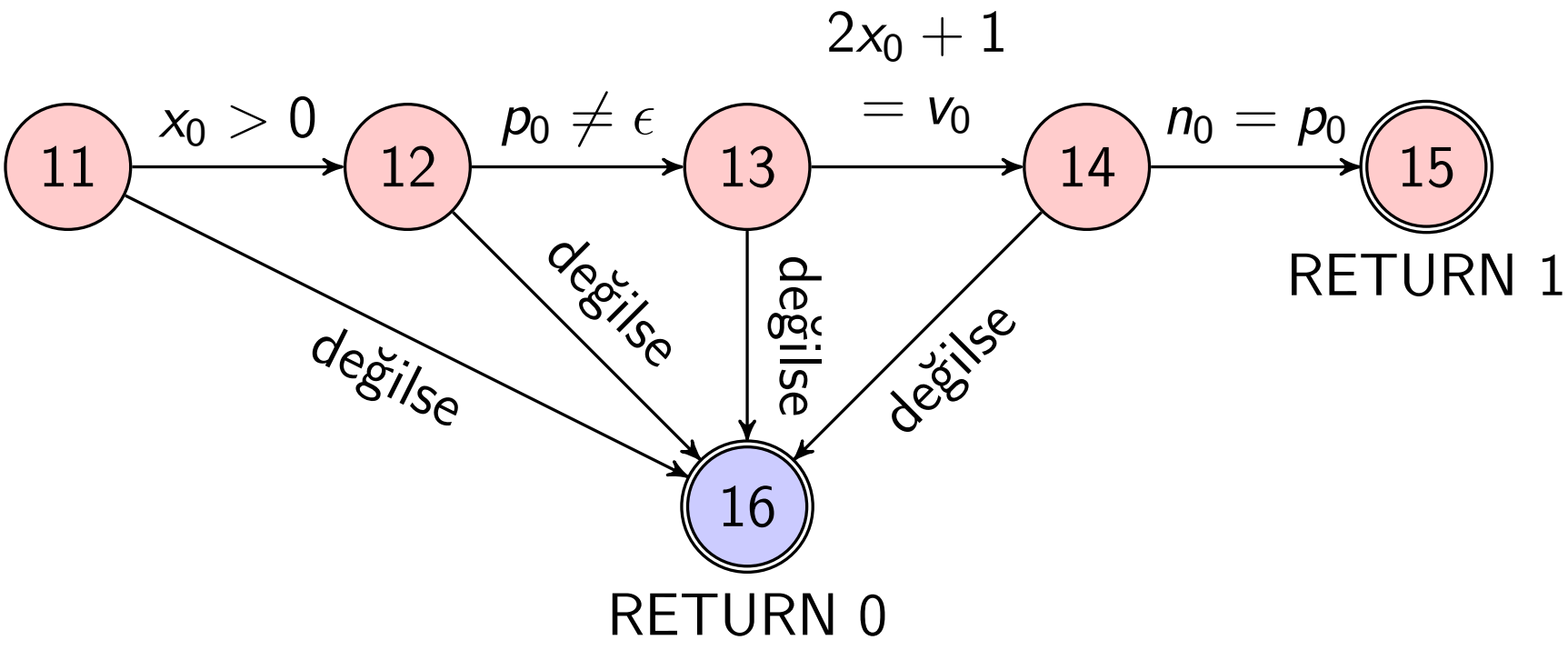
Konkolik İcra Örneği

Programı çalıştır ve sembolik kısıtları topla



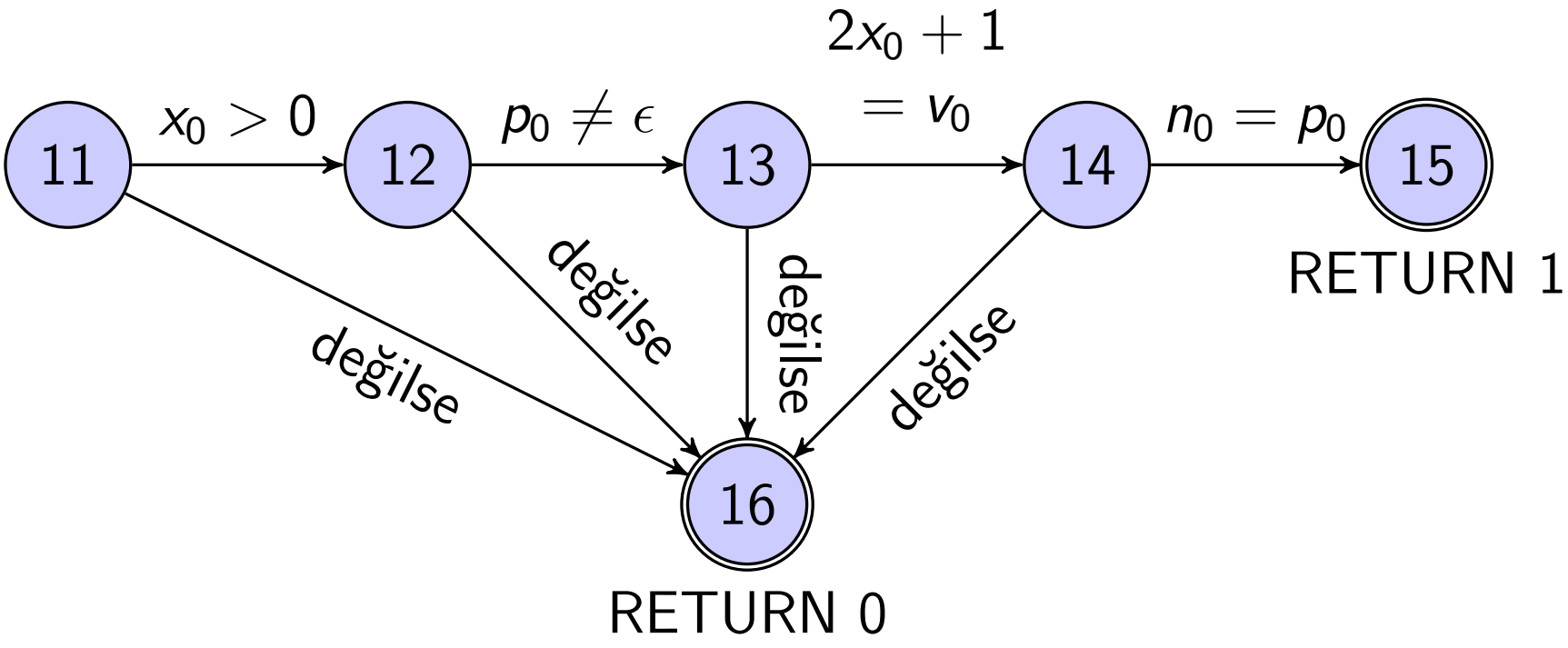
Konkolik İcra Örneği

$$\pi_4 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 = n_0\}$$



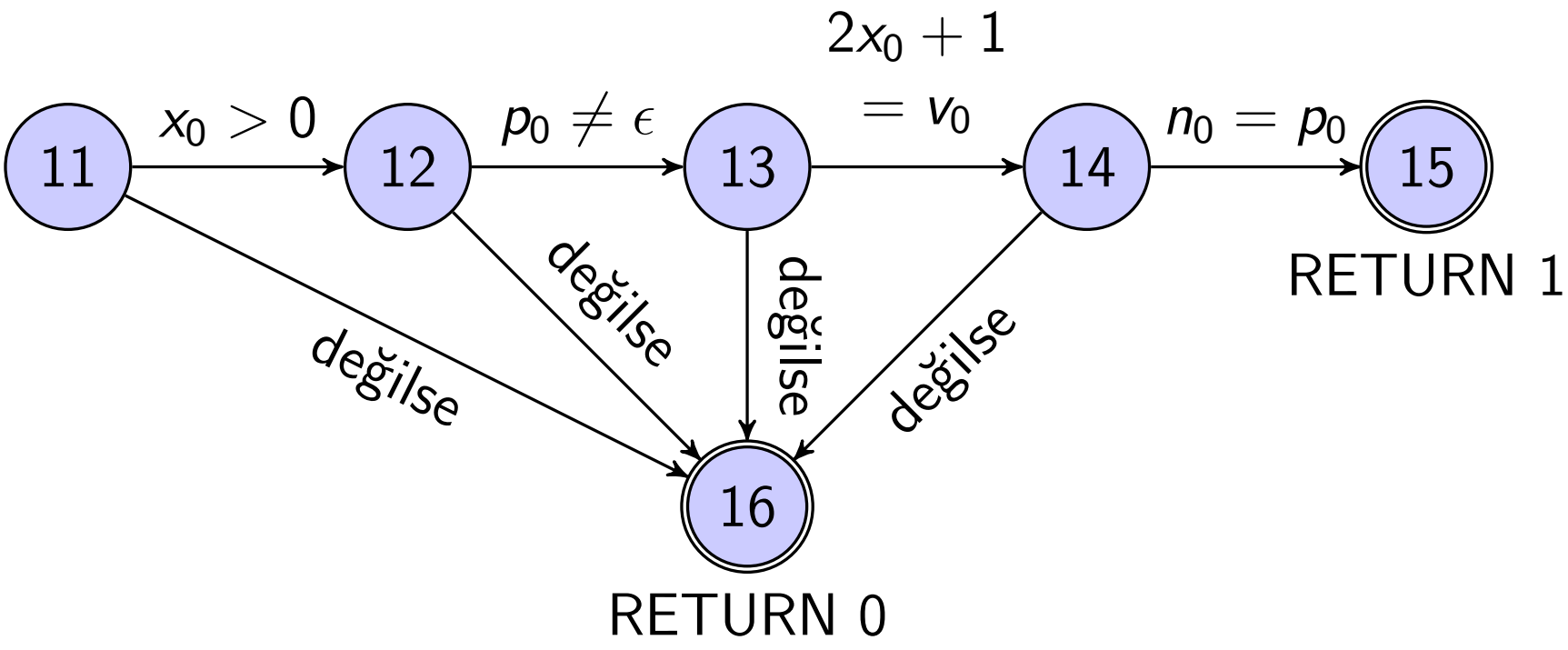
Konkolik İcra Örneği

π_4 kısıtının **denenmemiş** son koşulunu tes çevir.



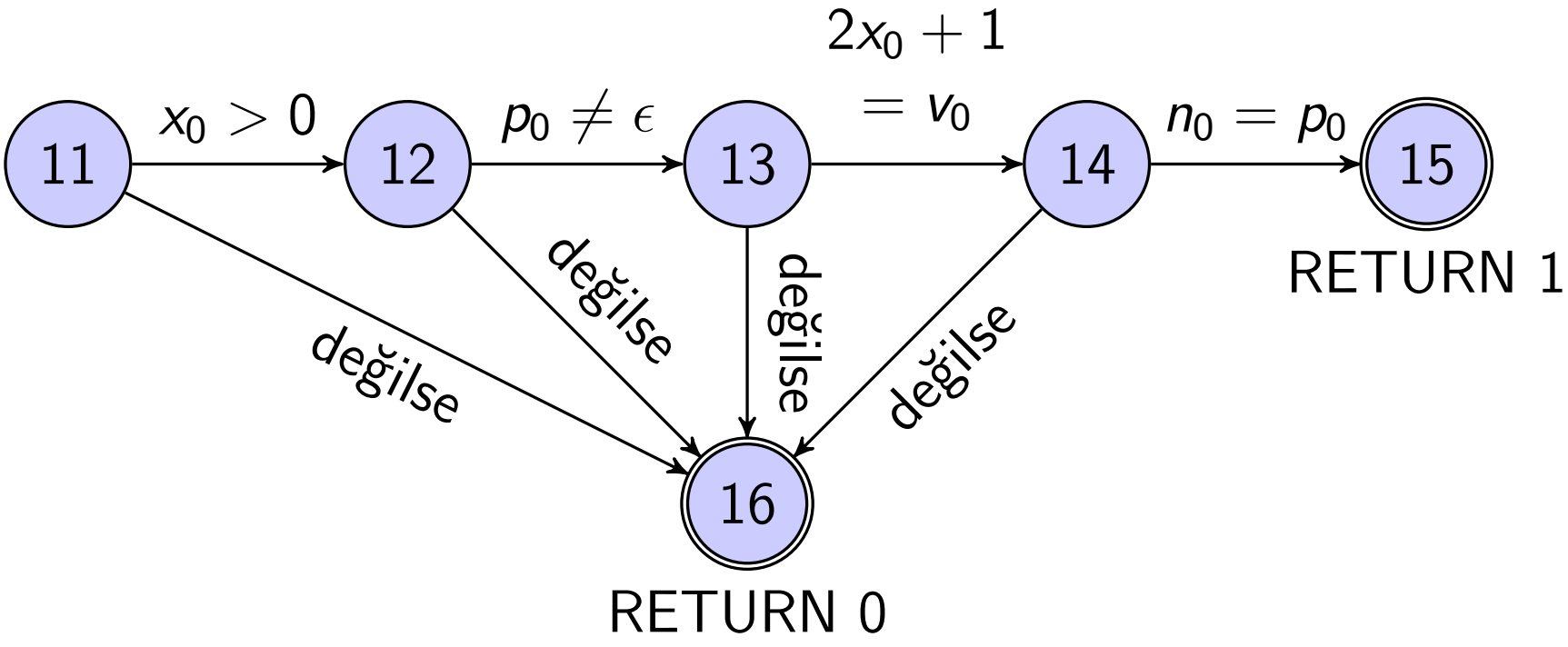
Konkolik İcra Örneği

$\pi_5 = \{x_0 > 0, p_0 = \epsilon\}$



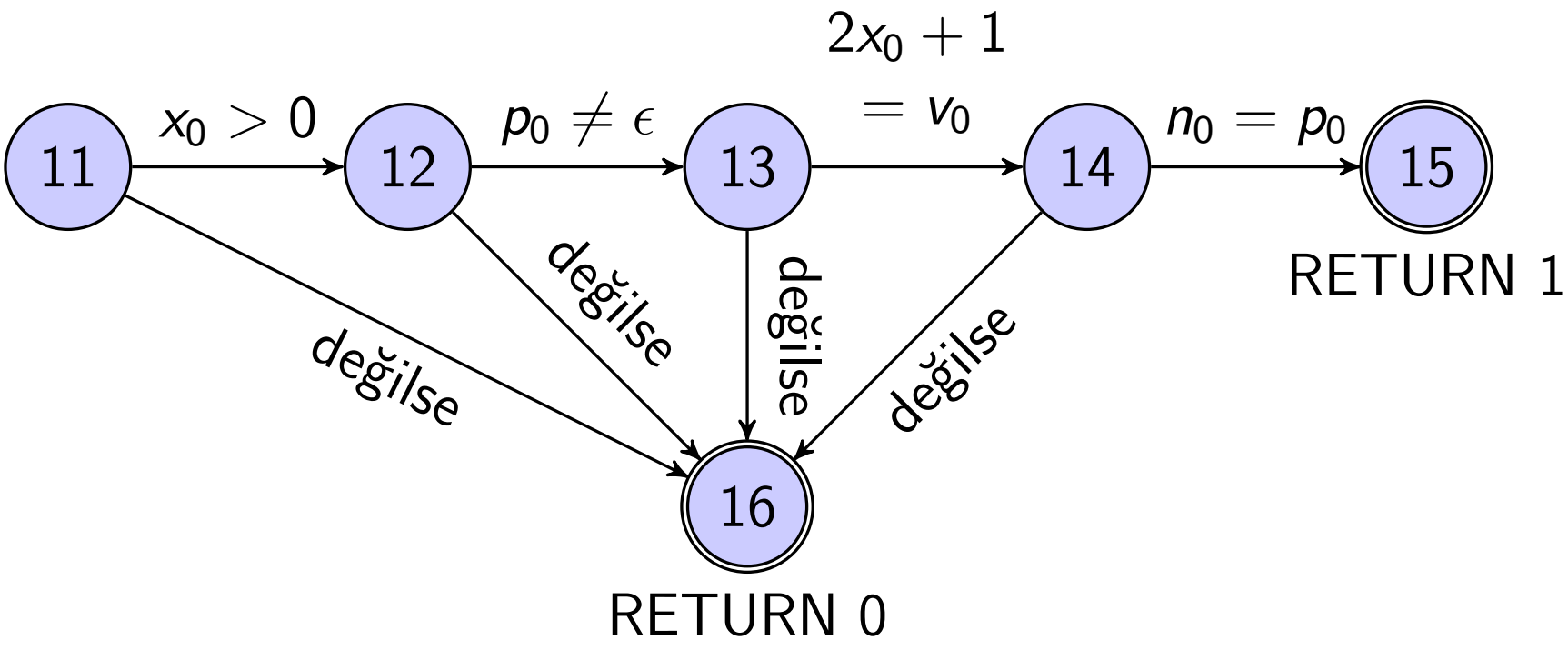
Konkolik İcra Örneği

ÇÖZ($\pi_5 = \{x_0 > 0, p_0 = \epsilon\}$)



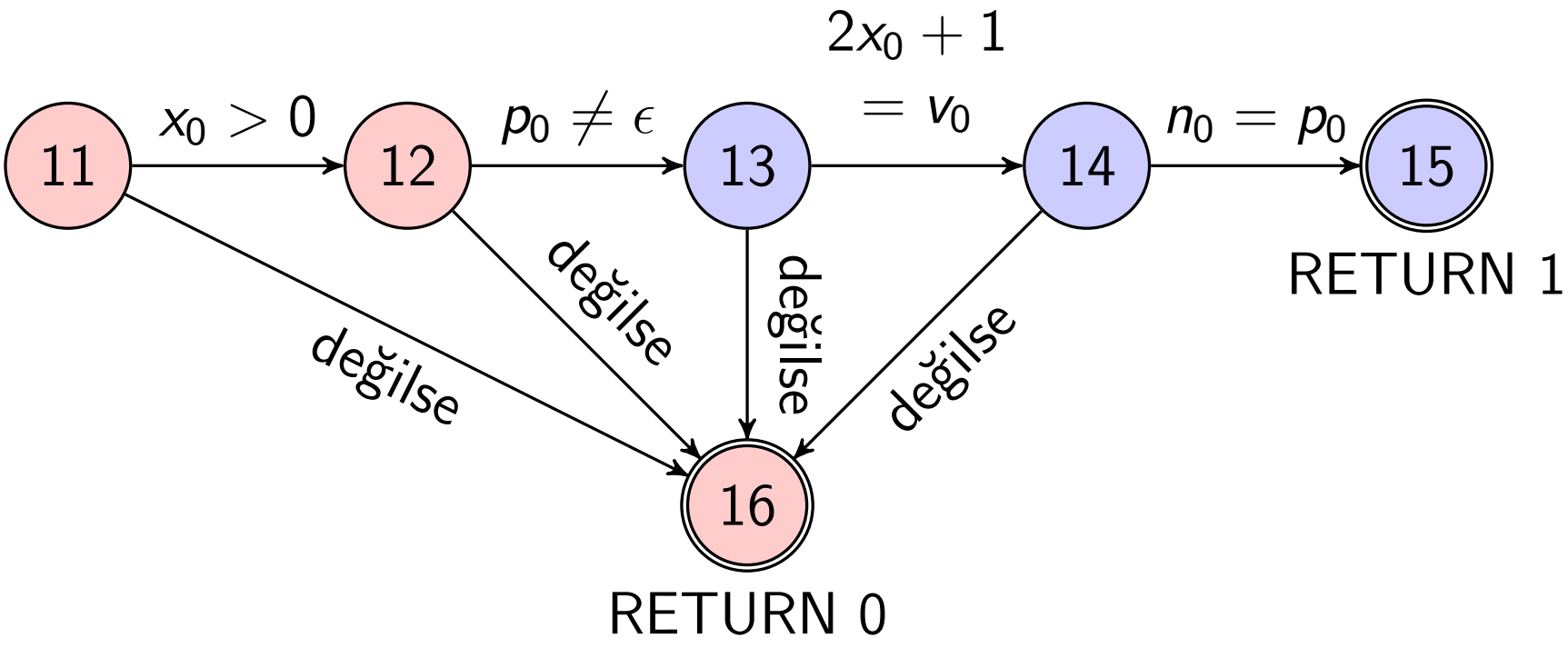
Konkolik İcra Örneği

$= [x_0 = 7, p_0 = \epsilon, v_0 = 15, n_0 = \epsilon]$



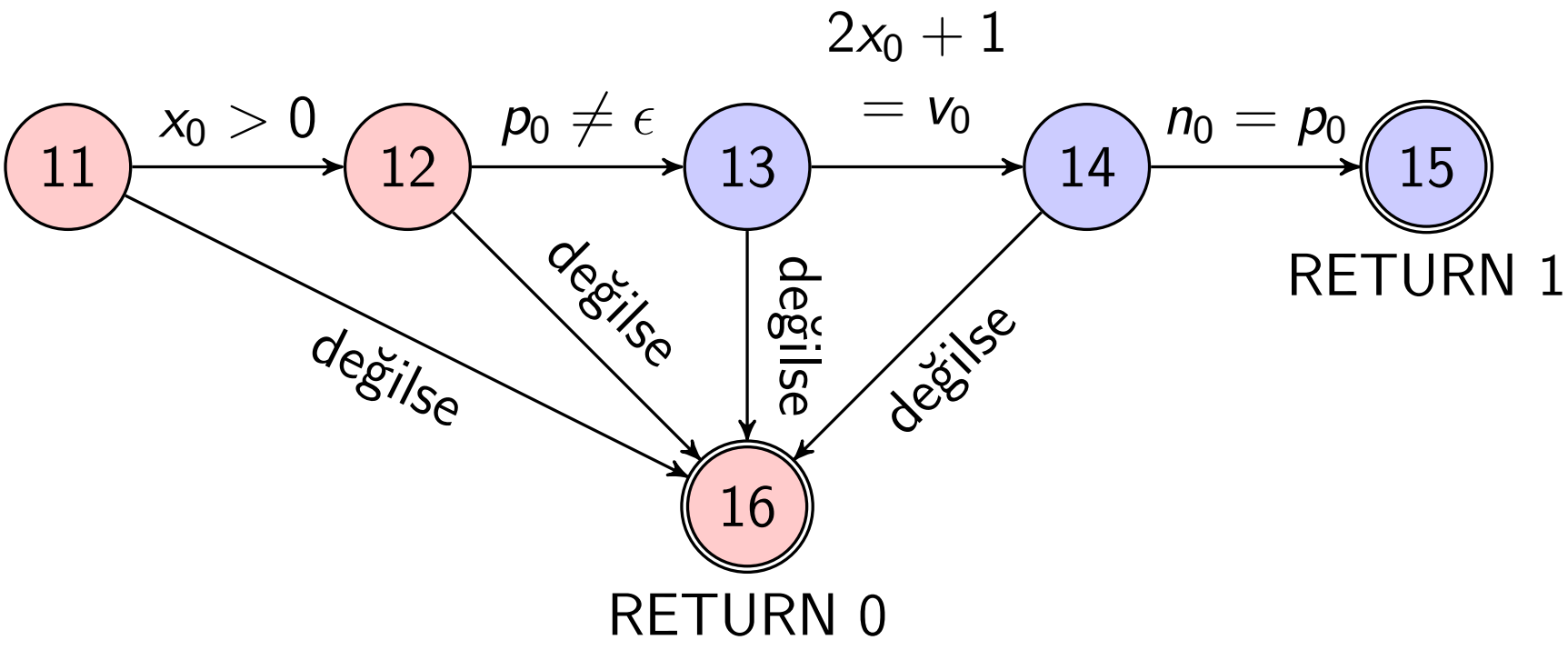
Konkolik İcra Örneği

Programı çalıştır ve sembolik kısıtları topla



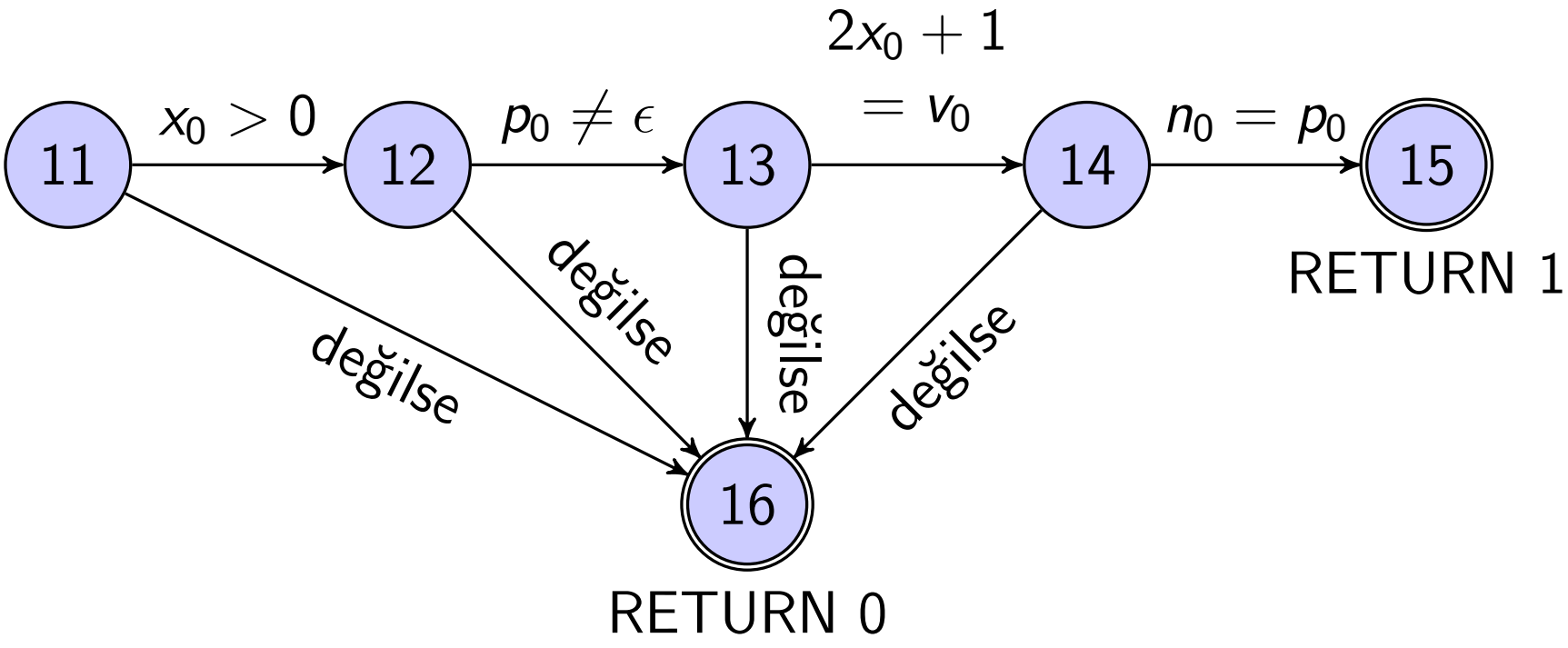
Konkolik İcra Örneği

$\pi_6 = \{x_0 > 0, p_0 = \epsilon\}$



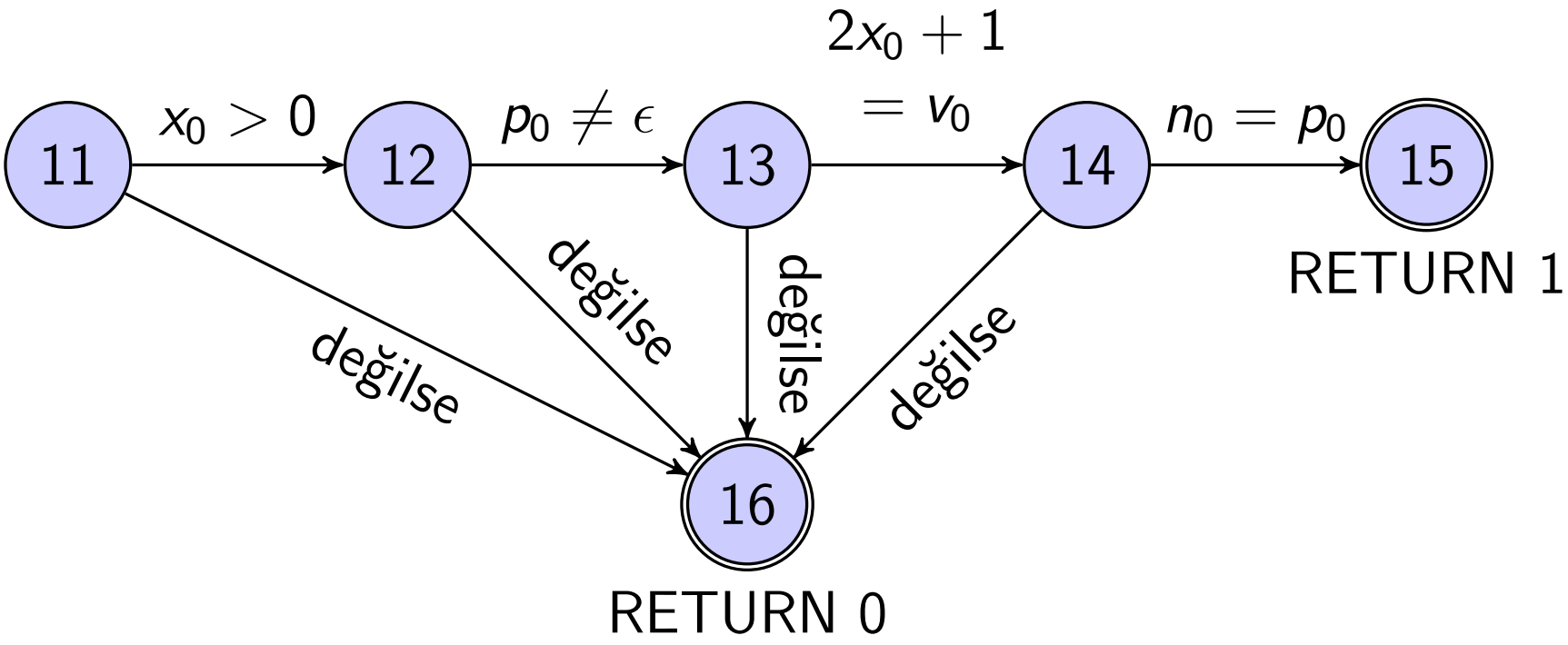
Konkolik İcra Örneği

π_6 kısıtının **denenmemiş** son koşulunu ters çevir.

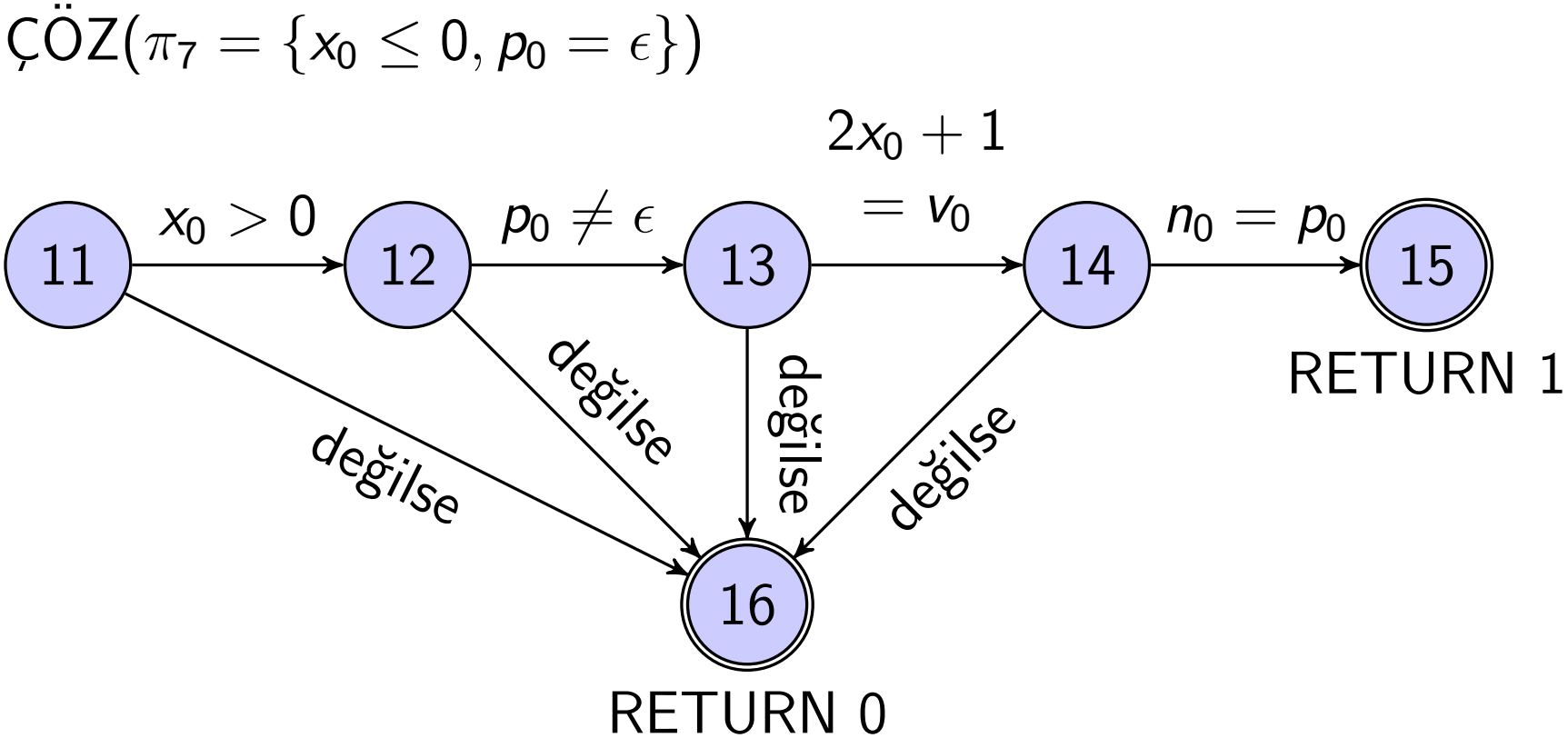


Konkolik İcra Örneği

$\pi_7 = \{x_0 \leq 0, p_0 = \epsilon\}$

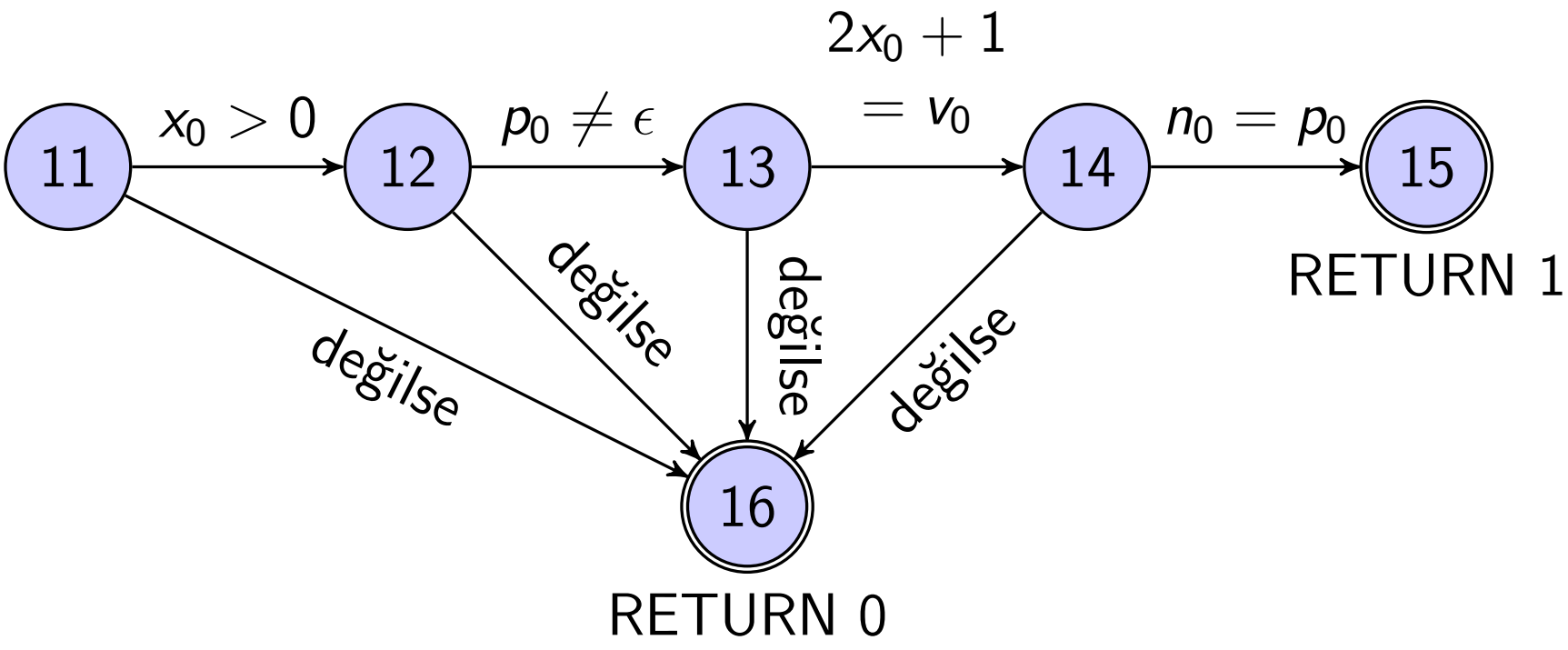


Konkolik İcra Örneği



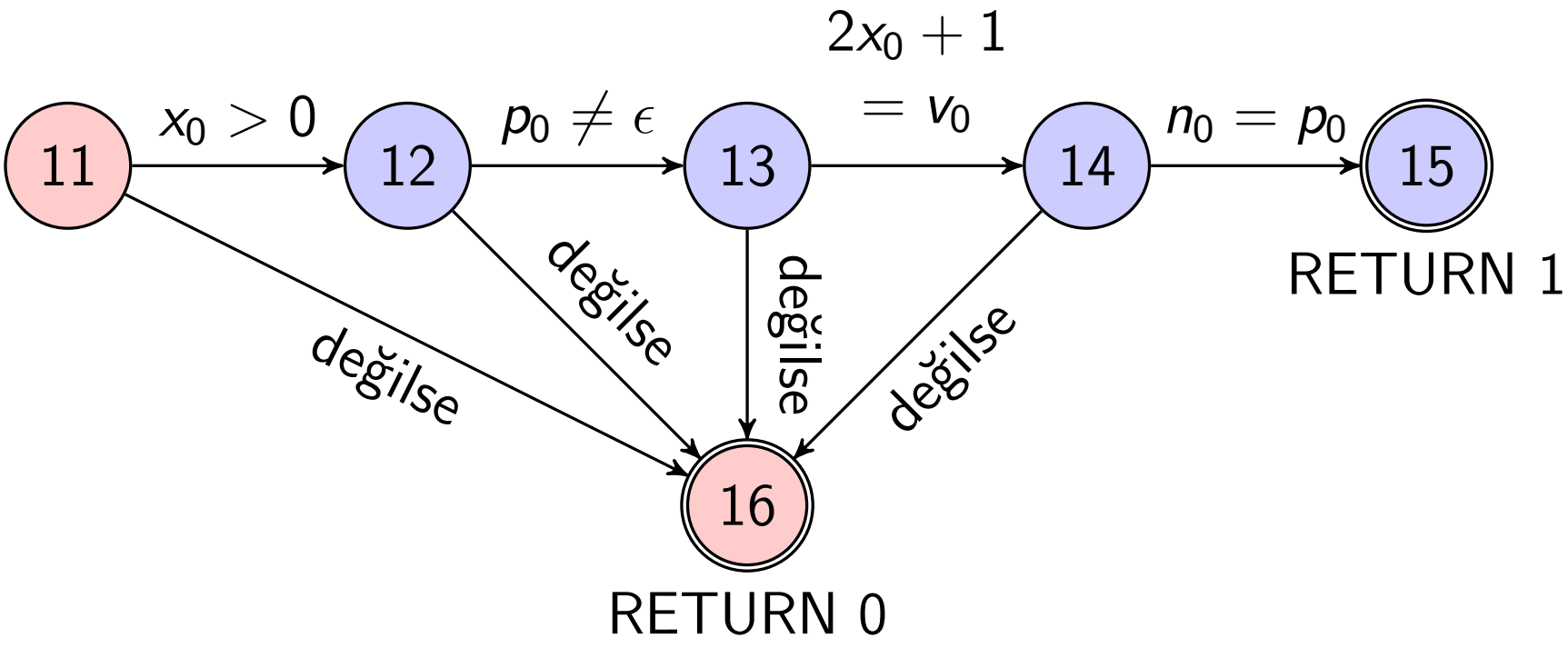
Konkolik İcra Örneği

$= [x_0 = 0, p_0 = \epsilon, v_0 = 15, n_0 = \epsilon]$



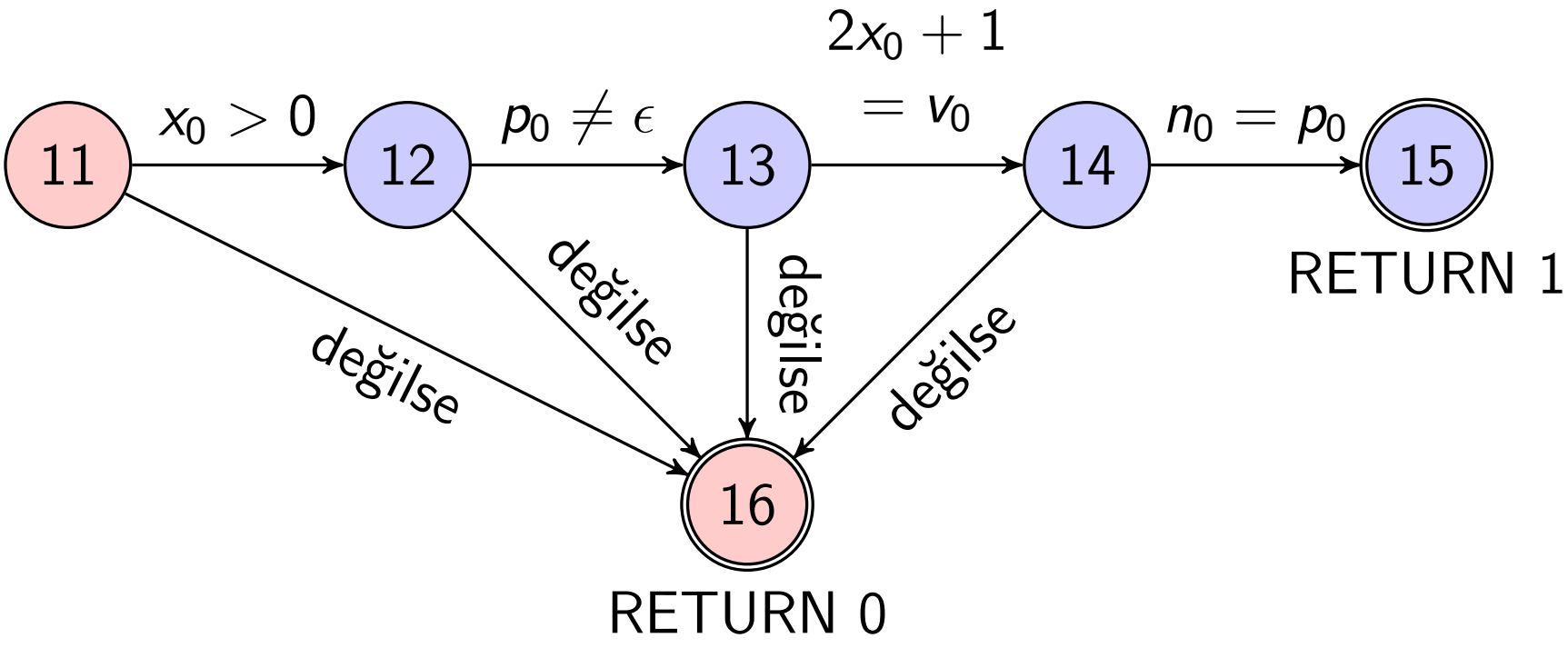
Konkolik İcra Örneği

Programı çalıştır ve sembolik kısıtları topla



Konkolik İcra Örneği

$\pi_8 = \{x_0 \leq 0\}$



Konkolik İcra Örneği

Sonuçlar

- Kısıt çözücü: 4 kere çağırıldı.
- Kısıt uzunluğu: Ortalama 2.75.
- İterasyon sayısı: 5.

Konkolik Test

Güncel Durum

- CREST [2] ile birkaç saniyede bir fonksiyonda 60 % dallanma kapsaması.
- Örnek sistemlerin test edilmesi en az **birkaç gün** almakta [7].

Darboğazlar

- Yol patlaması
- Yol sapması
- Kısıt çözümü

Konkolik Test

Genel İyileştirme Stratejileri

- Sadece belli hataları hedef almak (yol patlamasına karşı).
- Birimin çalışma yollarının uzunluğuna bir üst limit koymak (yol patlamasına karşı).
- Önceden toplanmış statik verileri testi yönlendirmek için kullanmak (kontrol-akış grafiği, CREST).
- Kısıt çözücünün yükünü optimizasyonlarla hafifletmek (kısıt çözümü).

Kısmi Yol Kısıtları Kullanan Konkolik Test

Tam Yol Kısıtı

Bir program yolunun çalıştırılabilmesi için gerekli sembolik değişkenler üzerine tanımlanmış bütün koşullar.

Kısmi Yol Kısıtı

Tam yol kısıtının herhangi bir alt kümesi.

- Küçük alt küme $\Rightarrow$ Yol Sapması
- Büyük alt küme $\Rightarrow$ Uzun Süreli Kısıt Çözümü

Kısmi Yol Kısıtları Kullanan Konkolik Test

Örnek

- a, b ve c program değişkenleri olsun.
- İstenilen noktaya ancak $a > b$, $a > c$ ve $b > c$ iken ulaşıldığı bilgisi elimize gelsin.
- Bu noktaya sırf $a > b$ ve $b > c$ yi sağlayan herhangi bir örnekle ulaşılabilir.
- Ama sadece $a > b$ yi sağlayan bir örnek bu noktaya **ulaşabilir de ulaşmayabilir de**.

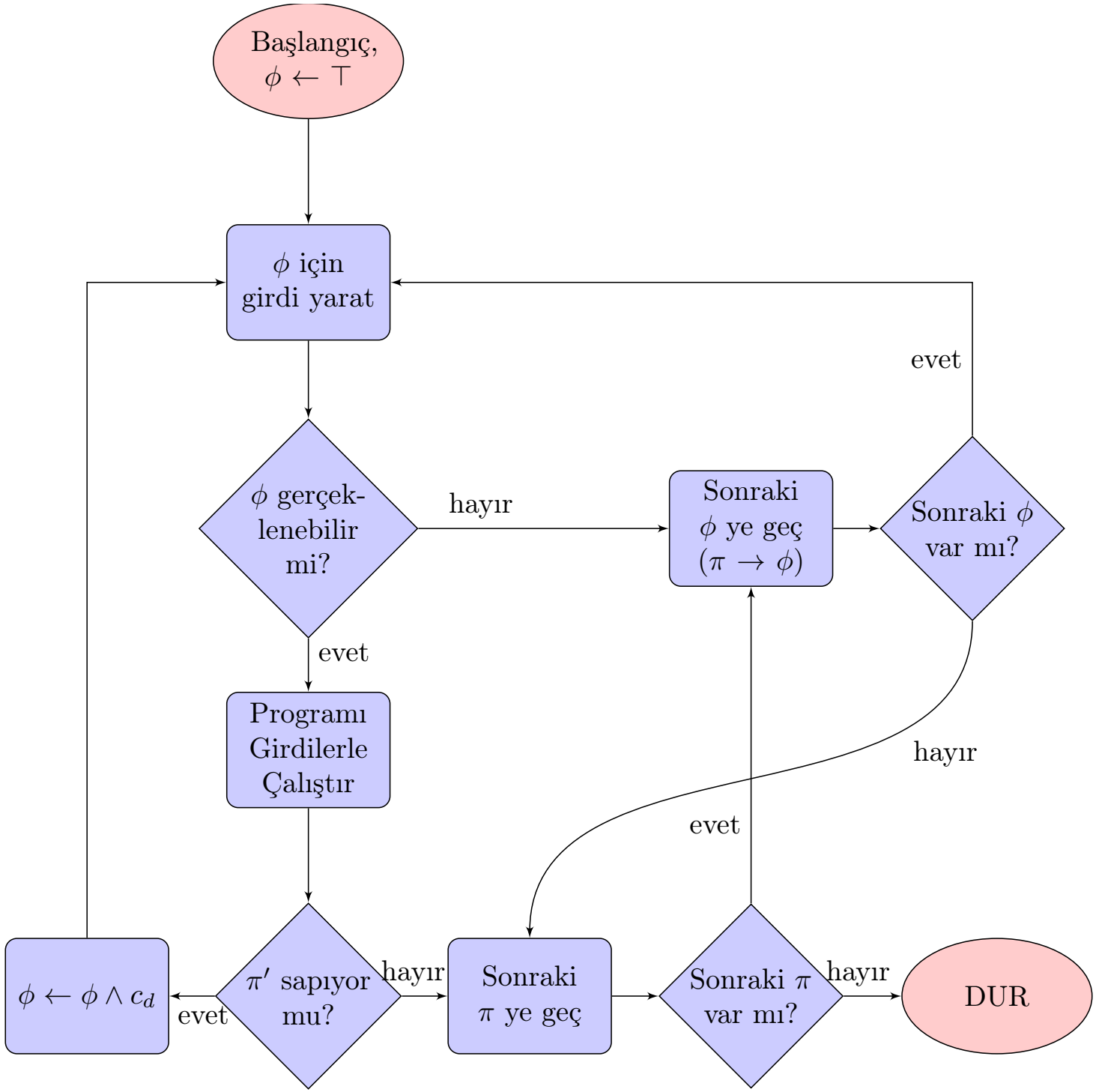
Büyüyen Kısmi Yol Kısıtları

Genel Strateji

1. Gerekli yol kısıtının sadece son koşuluyla başla.
2. Bu kısmi kısıt için test girdileri üret (kısıt çözücü).
3. Yol sapması nedenini (dallanma koşulu) öğren.
4. Bu koşulu kısmi kısıta ekle ve (1) e git.

Notlar

- Kısmi kısıtın bir çözümü bulunmayabilir (Program yolu **ölü**.)
- Kısmi kısıt yol sapmasına neden olmayabilir (Test başarılı!)


Başlangıç,
$\phi \leftarrow \top$
ϕ için
girdi yarat
ϕ gerçek-
lenebilir
mi?
hayır
Sonraki
ϕ ye geç
$(\pi \rightarrow \phi)$
Sonraki ϕ
var mı?
evet
evet
Programı
Girdilerle
Çalıştır
hayır
evet
$\phi \leftarrow \phi \wedge c_d$
evet
π' sapıyor
mu?
hayır
Sonraki
π ye geç
Sonraki π
var mı?
hayır
DUR

Büyüyen Kısmi Yol Kısıtları İle İcra

1.Adım: Sembolik değişkenlerin el ile belirlenmesi.

```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

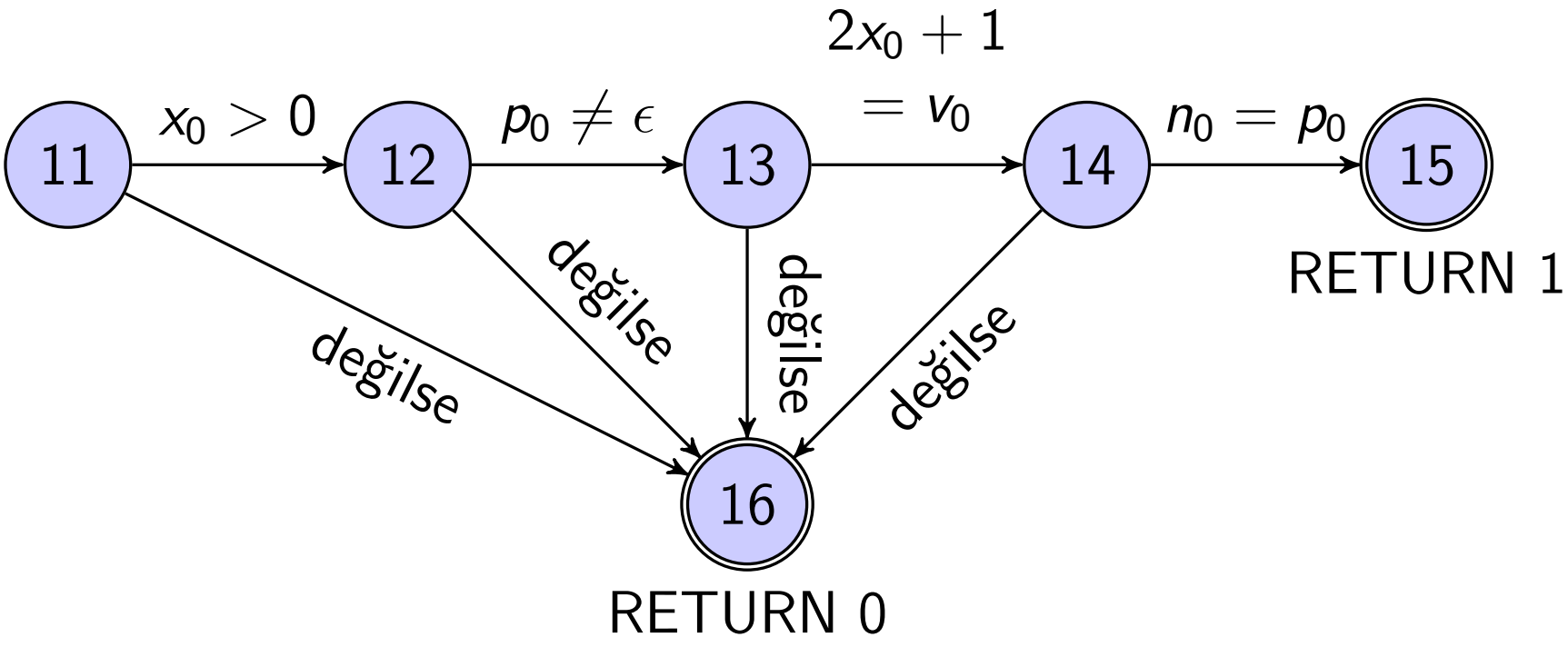
Büyüyen Kısmi Yol Kısıtları İle İcra

Değişkenler: p_0, v_0, n_0, x_0

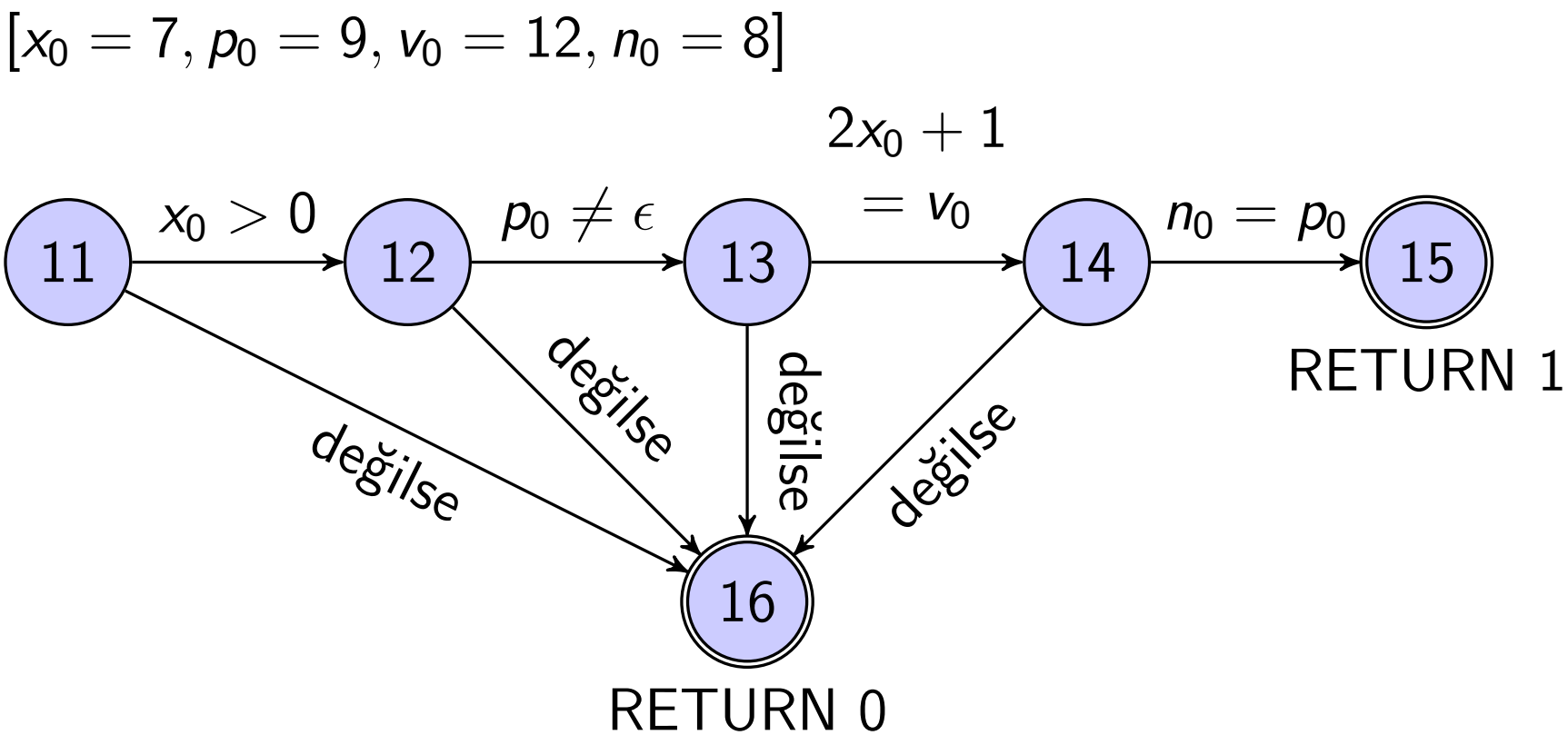
```
1  typedef struct cell {
2      int v;
3      struct cell * n;
4  } cell;
5
6  int f(int x) {
7      return 2 * x + 1;
8  }
9
10 int testme(cell * p, int x) {
11     if (x > 0)
12         if (p != NULL)
13             if (f(x) == p->v)
14                 if (p->n == p)
15                     return 1;
16     return 0;
17 }
```

Büyüyen Kısmi Yol Kısıtları İle İcra

2.Adım: Sembolik değişkenlere rastgele değerler ata.

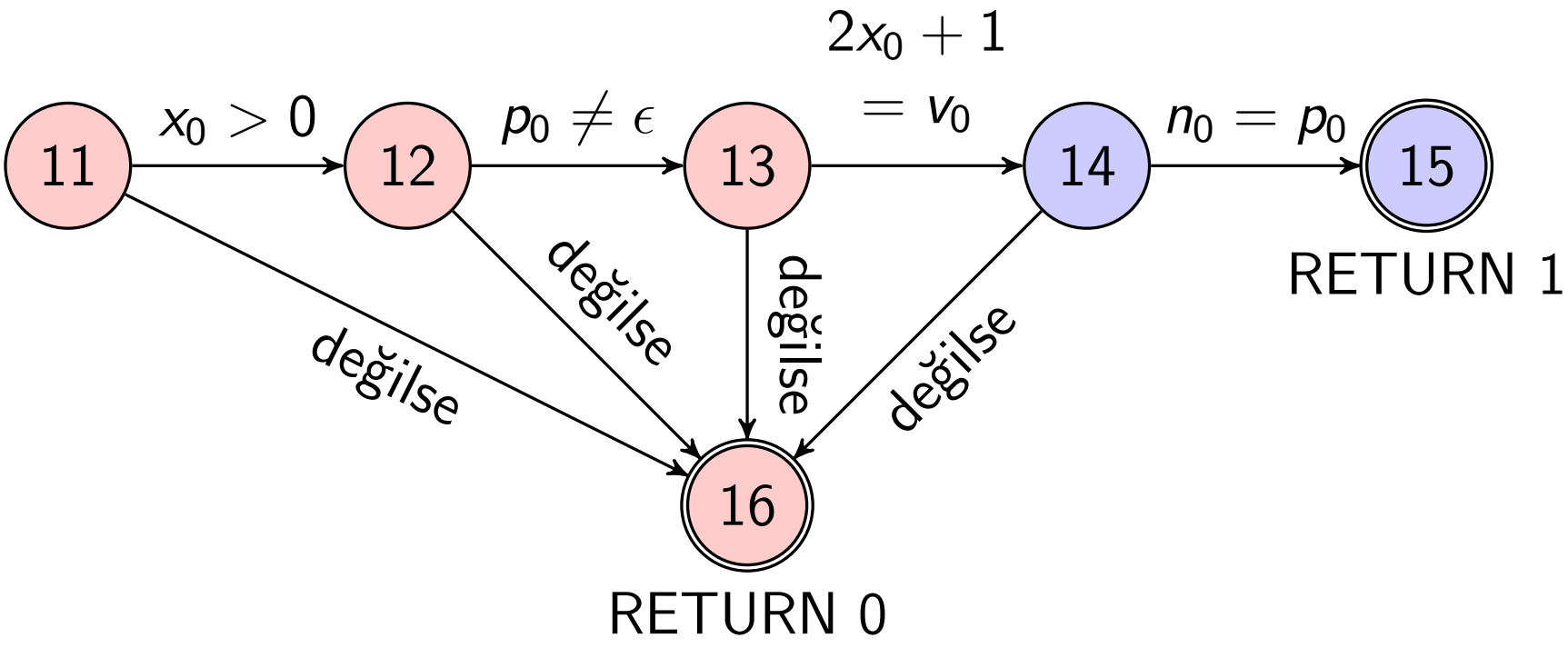


Büyüyen Kısmi Yol Kısıtları İle İcra



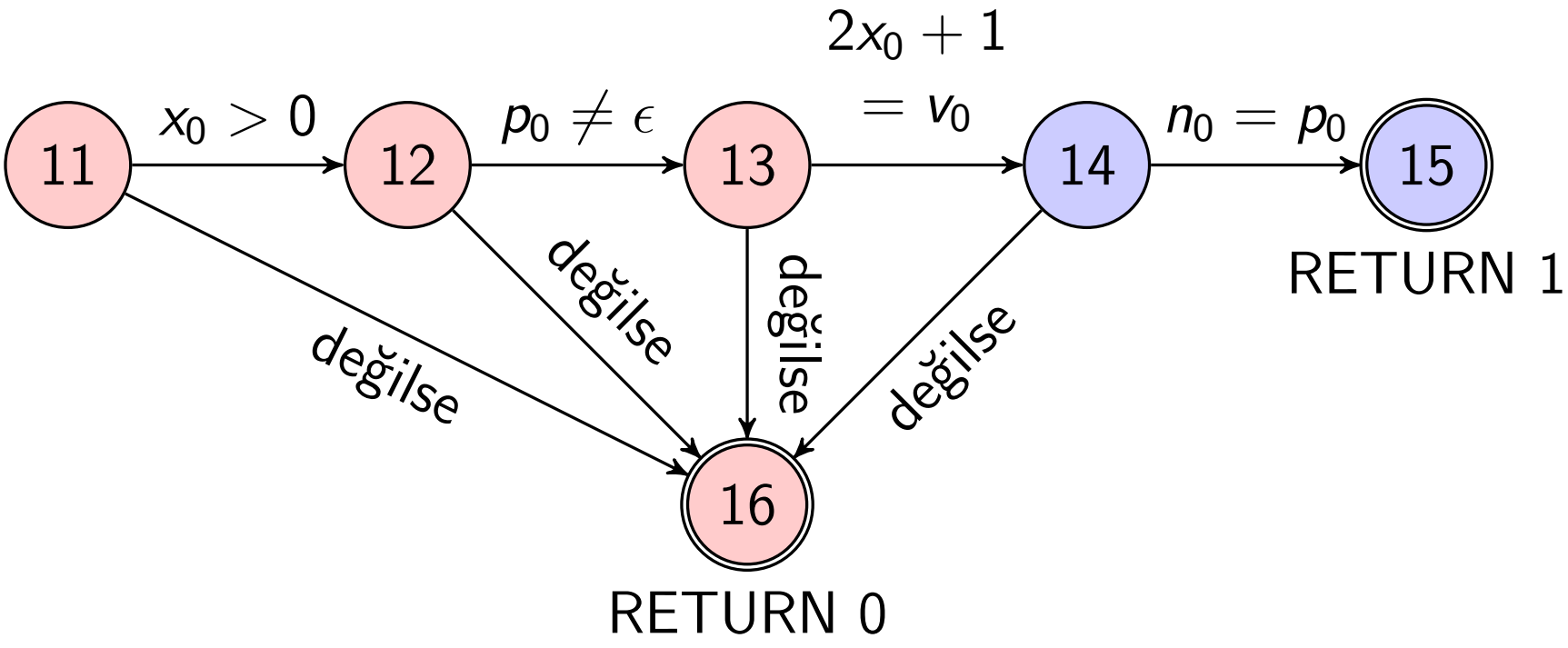
Büyüyen Kısmi Yol Kısıtları İle İcra

Programı çalıştır ve sembolik kısıtları topla



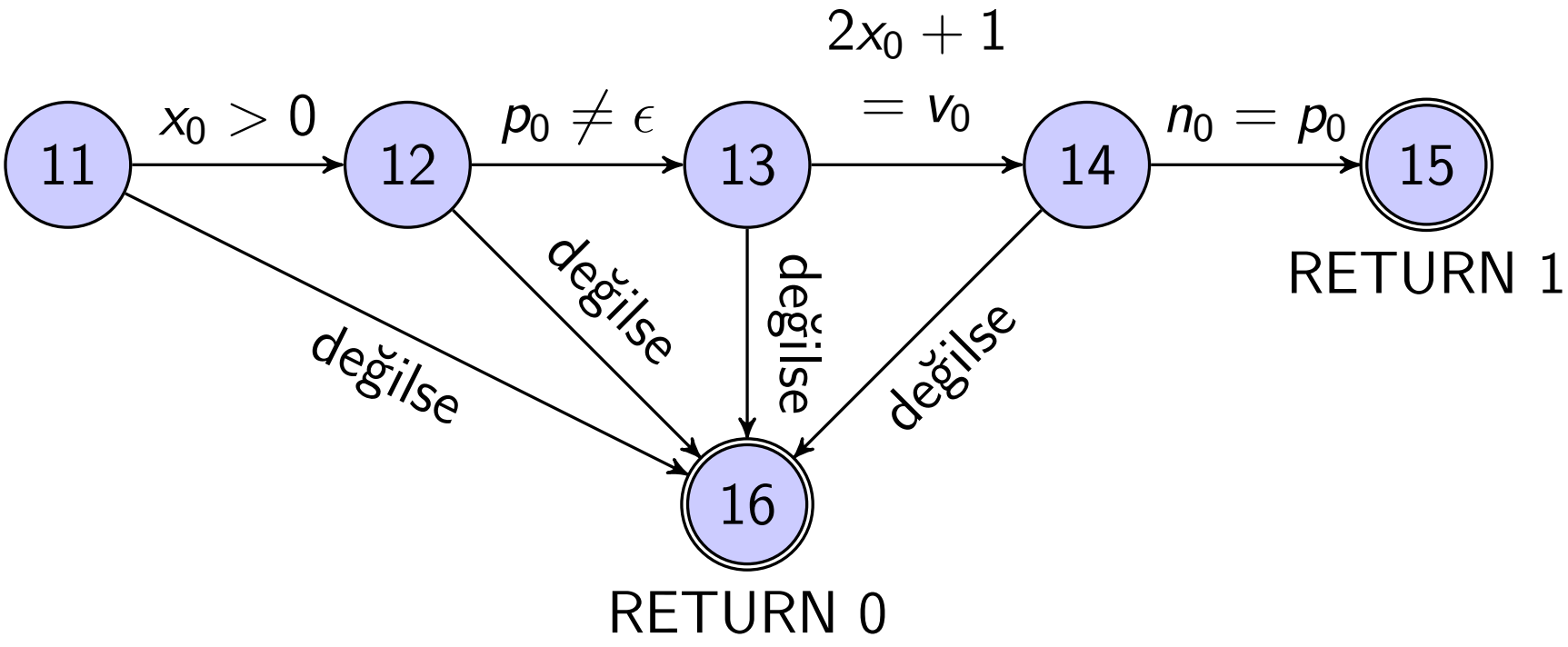
Büyüyen Kısmi Yol Kısıtları İle İcra

$\pi_0 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 \neq v_0\}$



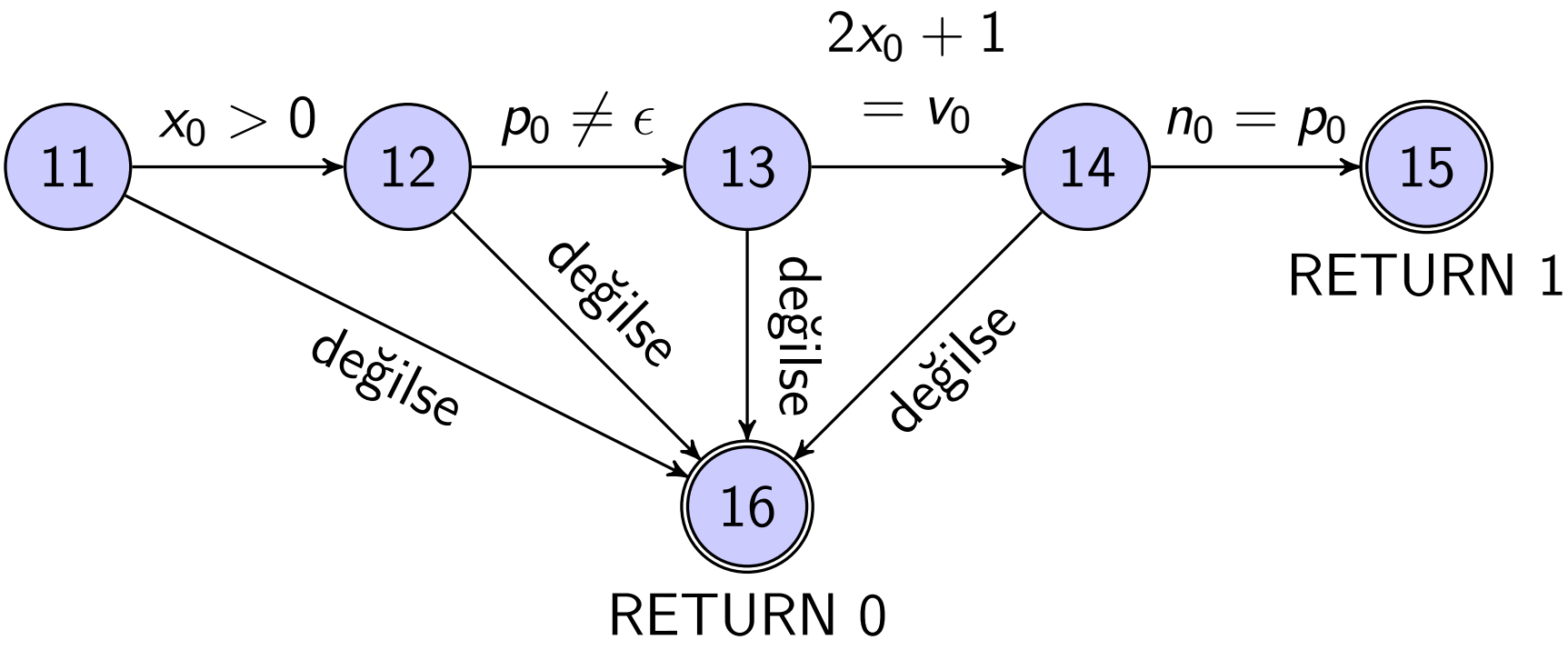
Büyüyen Kısmi Yol Kısıtları İle İcra

π_0 kısıtının son koşulunu tersine çevirerek yeni kısıt elde et



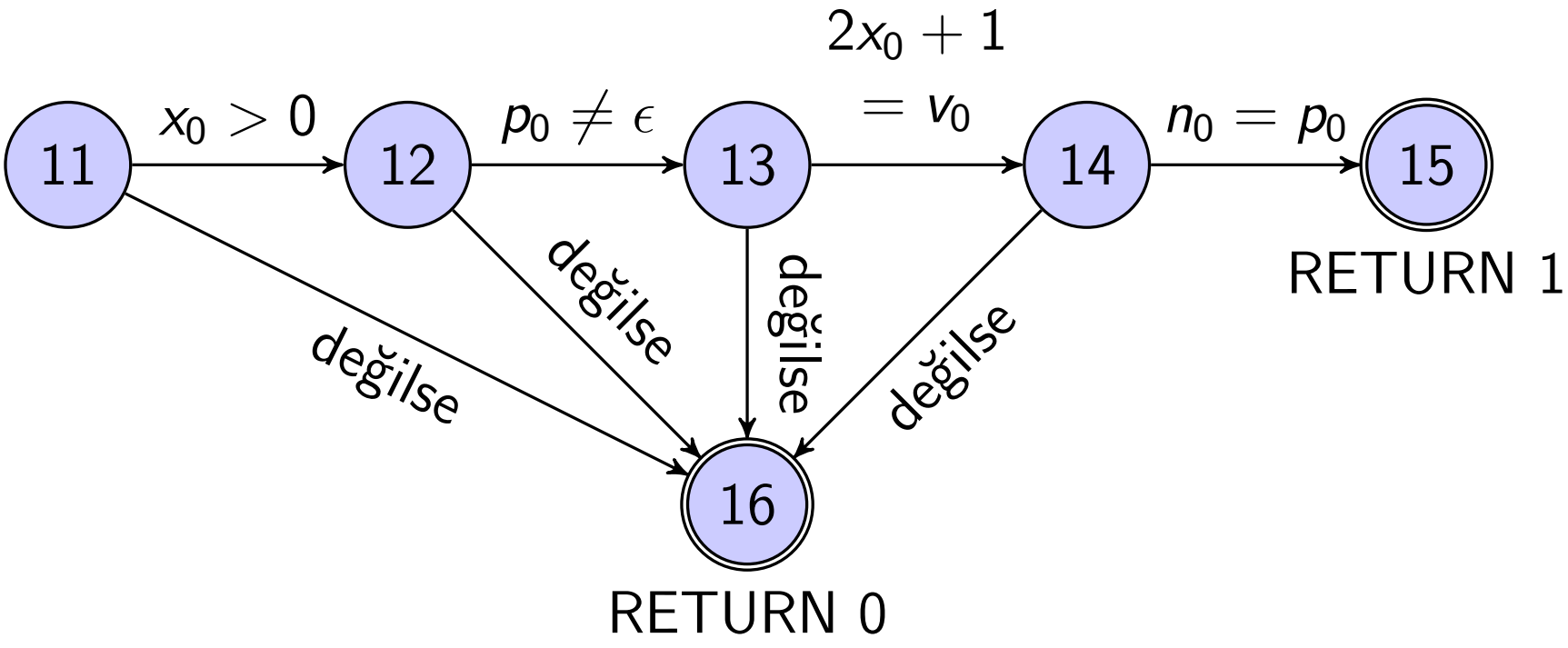
Büyüyen Kısmi Yol Kısıtları İle İcra

$\pi_1 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0\}$

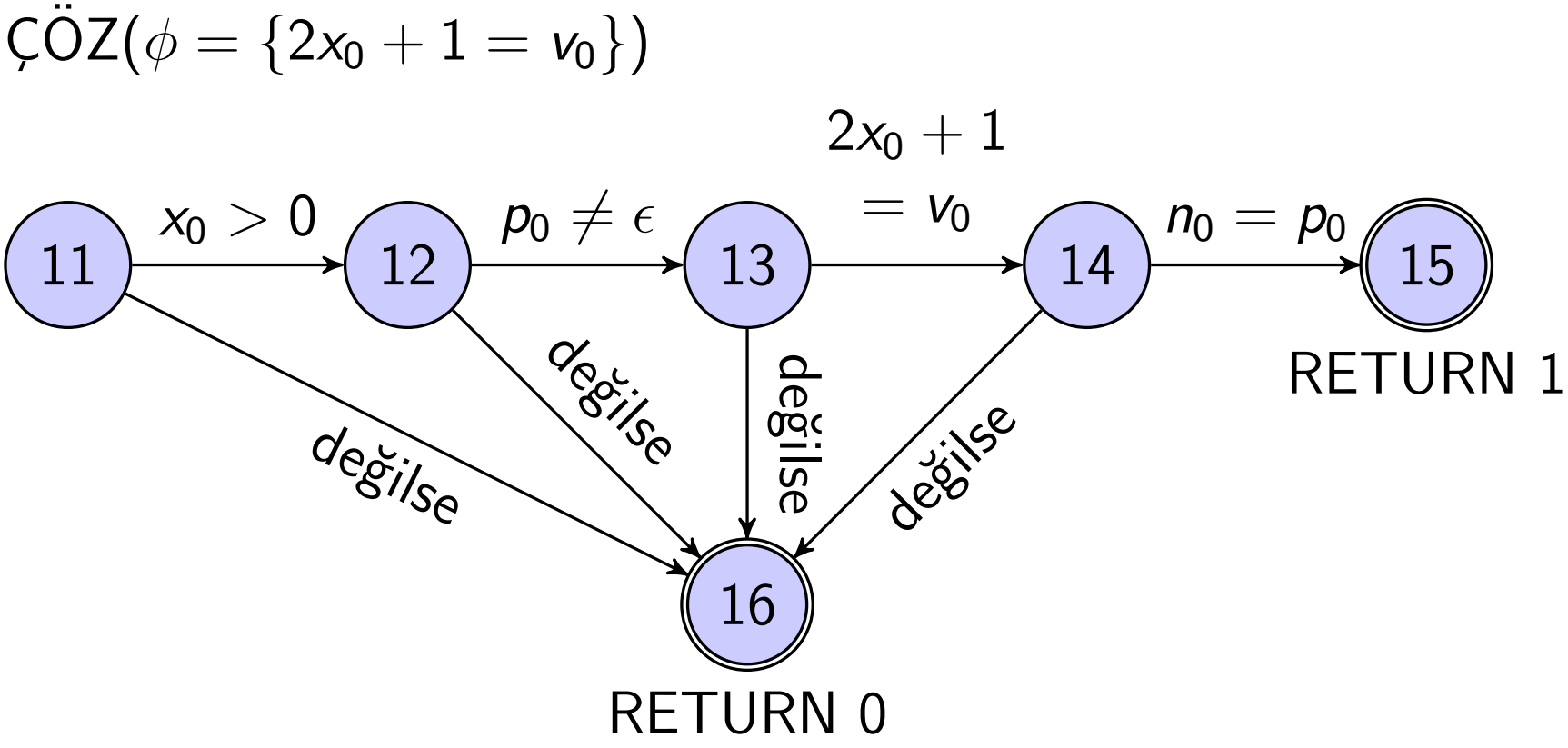


Büyüyen Kısmi Yol Kısıtları İle İcra

π_1 in sadece son koşulunu kısmi kısıt olarak al

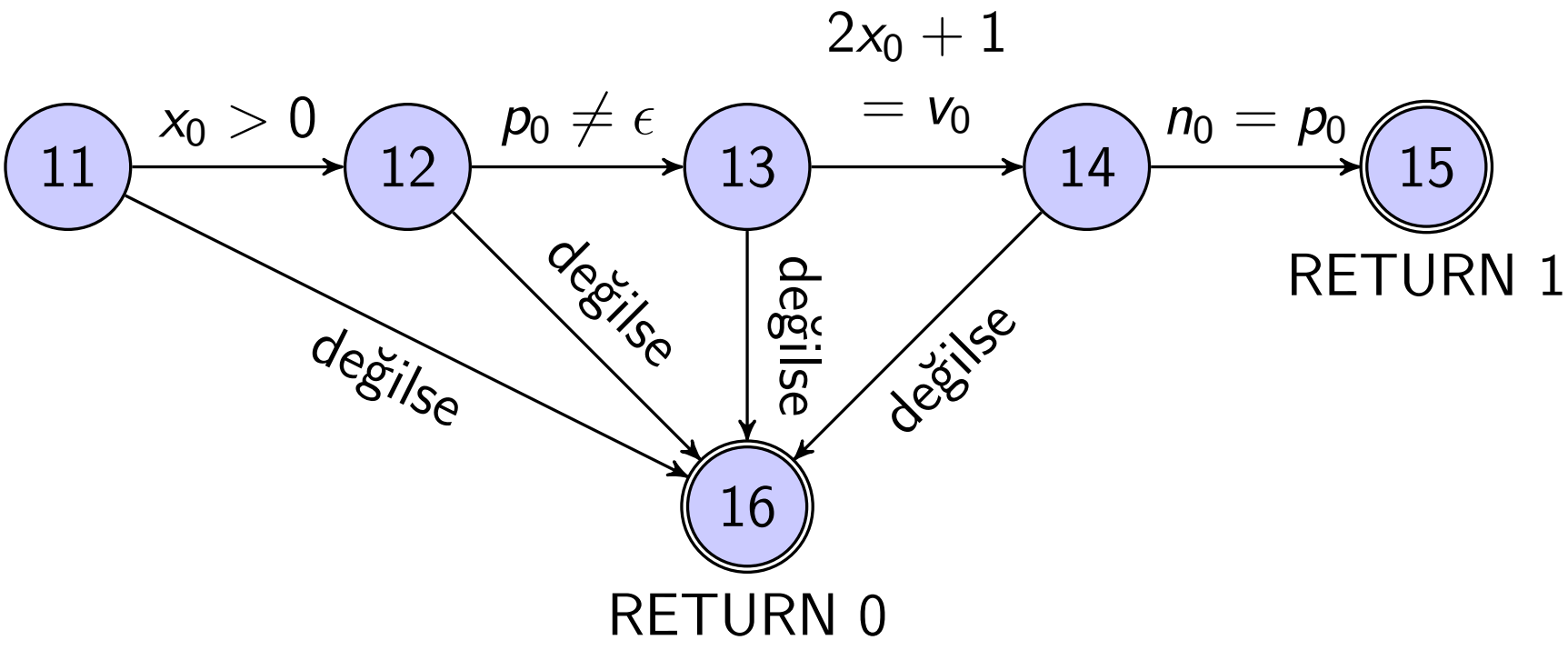


Büyüyen Kısmi Yol Kısıtları İle İcra



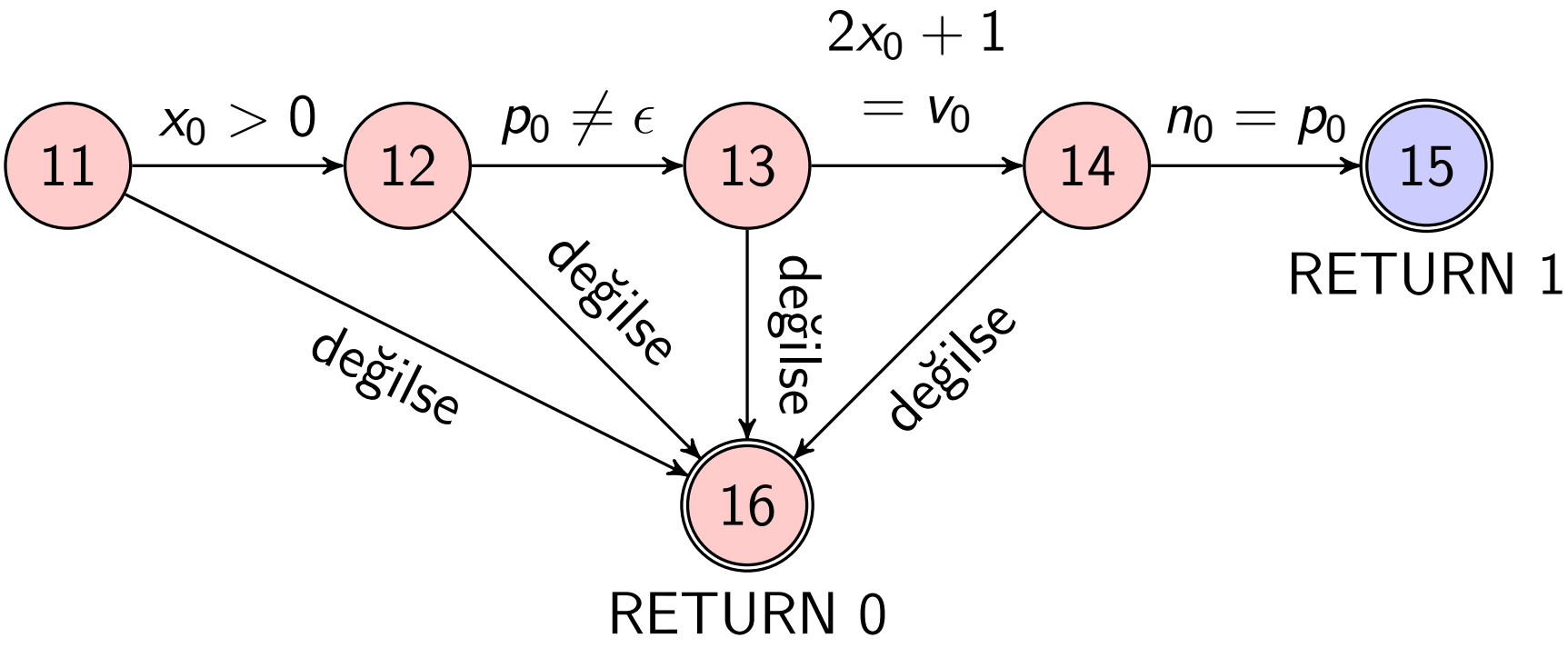
Büyüyen Kısmi Yol Kısıtları İle İcra

$= [x_0 = 7, p_0 = 9, v_0 = 15, n_0 = 8]$



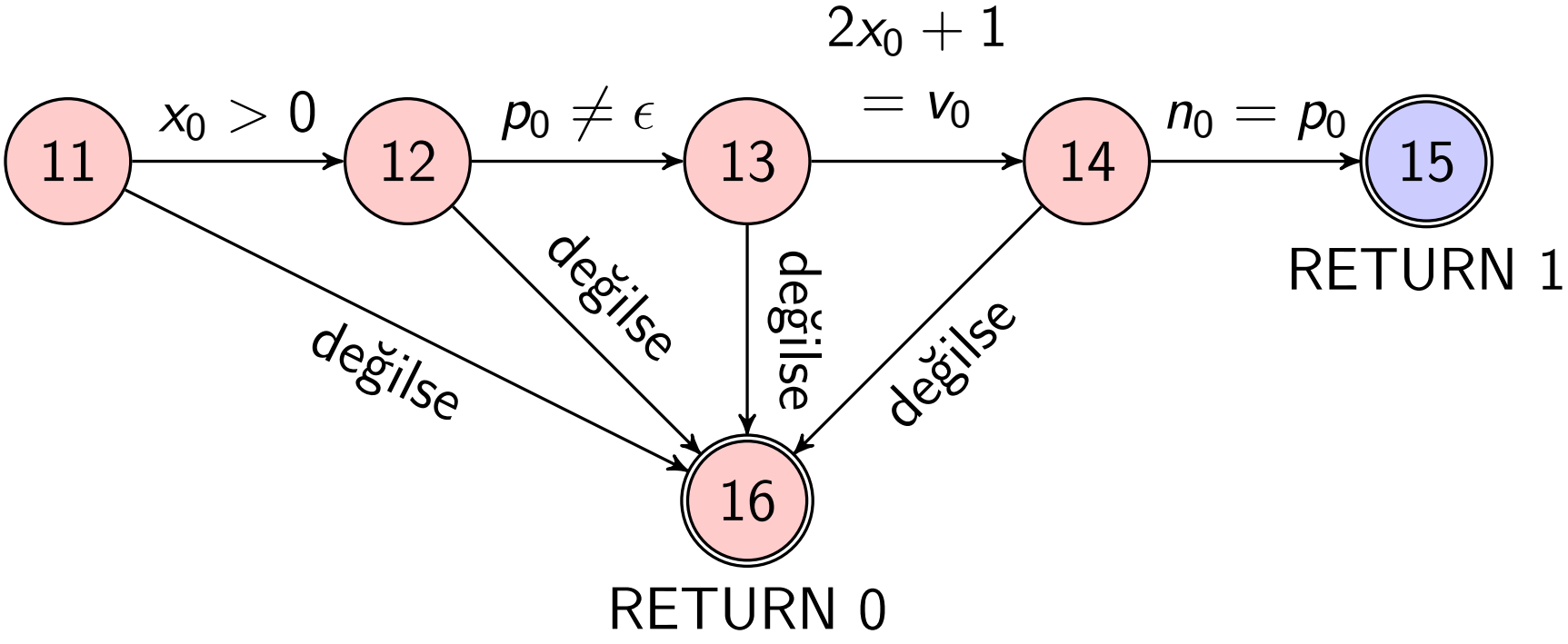
Büyüyen Kısmi Yol Kısıtları İle İcra

Programı çalıştır ve sembolik kısıtları topla



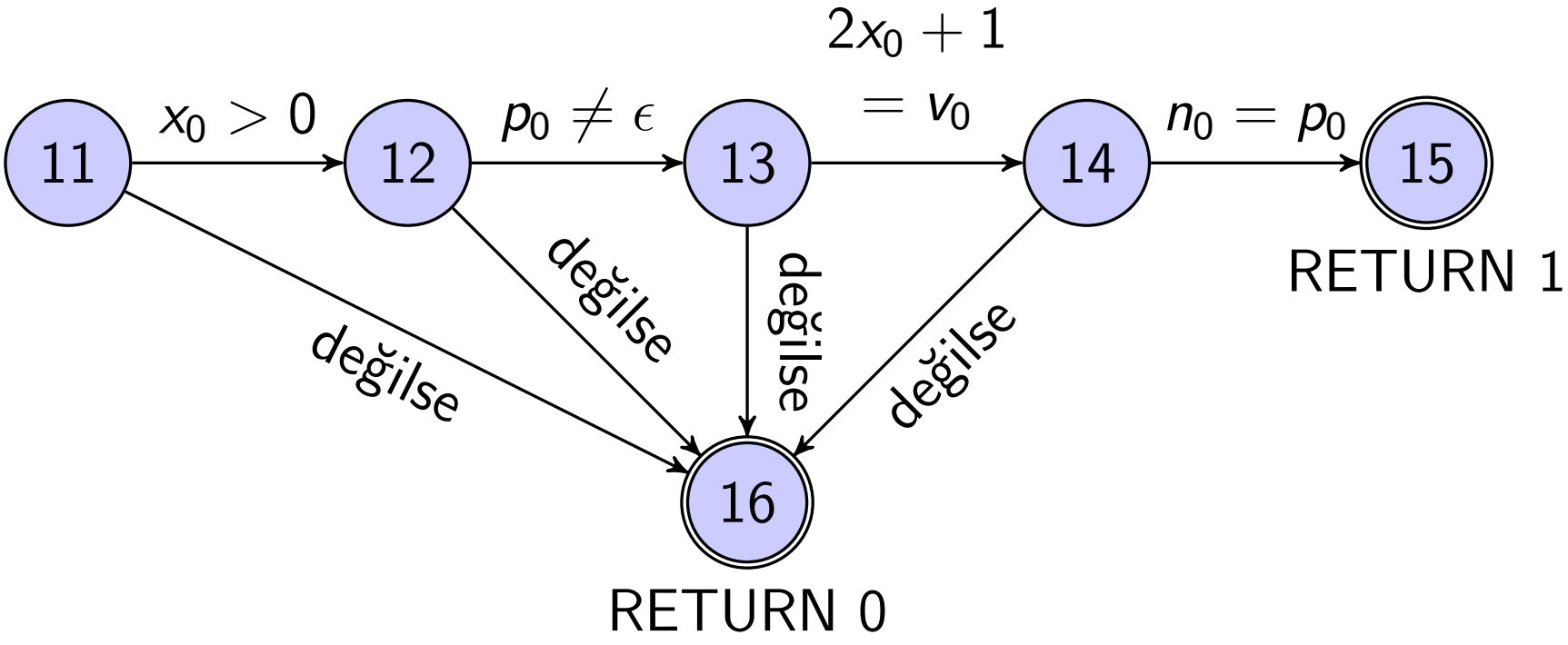
Büyüyen Kısmi Yol Kısıtları İle İcra

$$\pi_2 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 \neq n_0\}$$



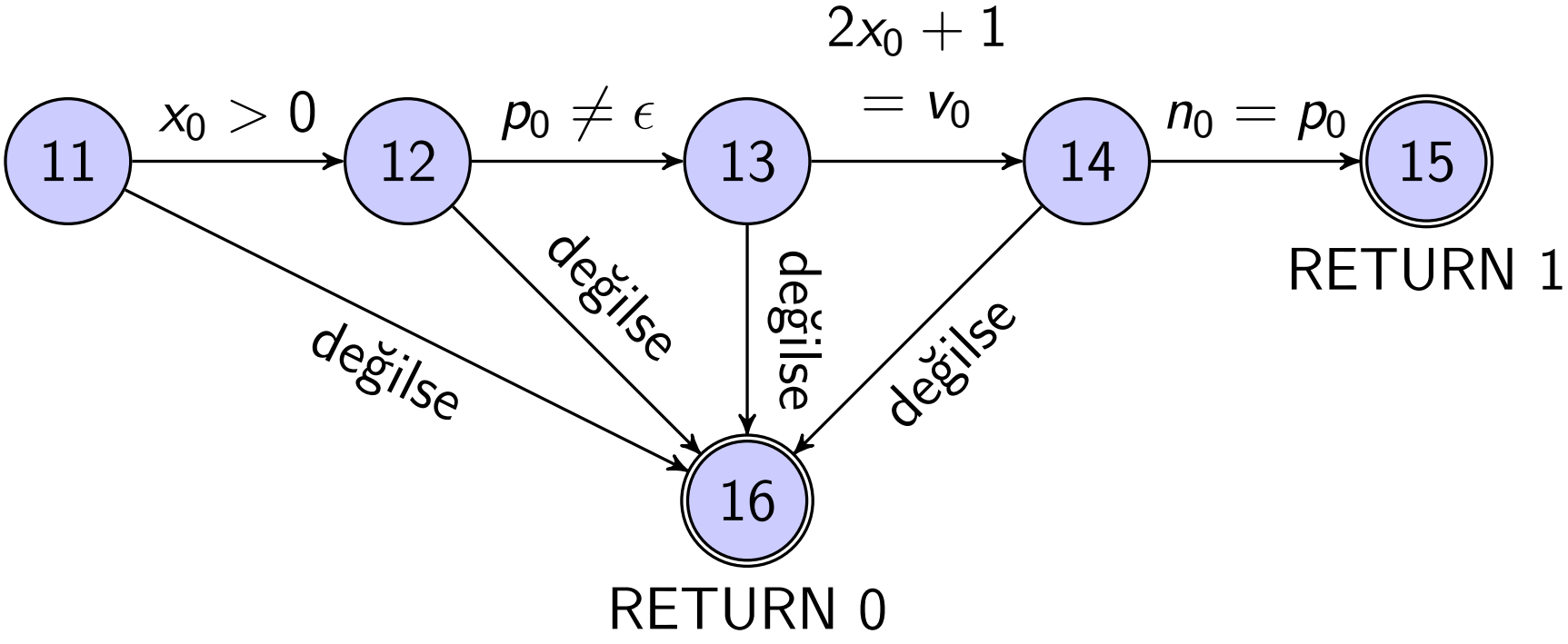
Büyüyen Kısmi Yol Kısıtları İle İcra

π_2 kısıtının son koşulunu tersine çevirerek yeni kısıt elde et



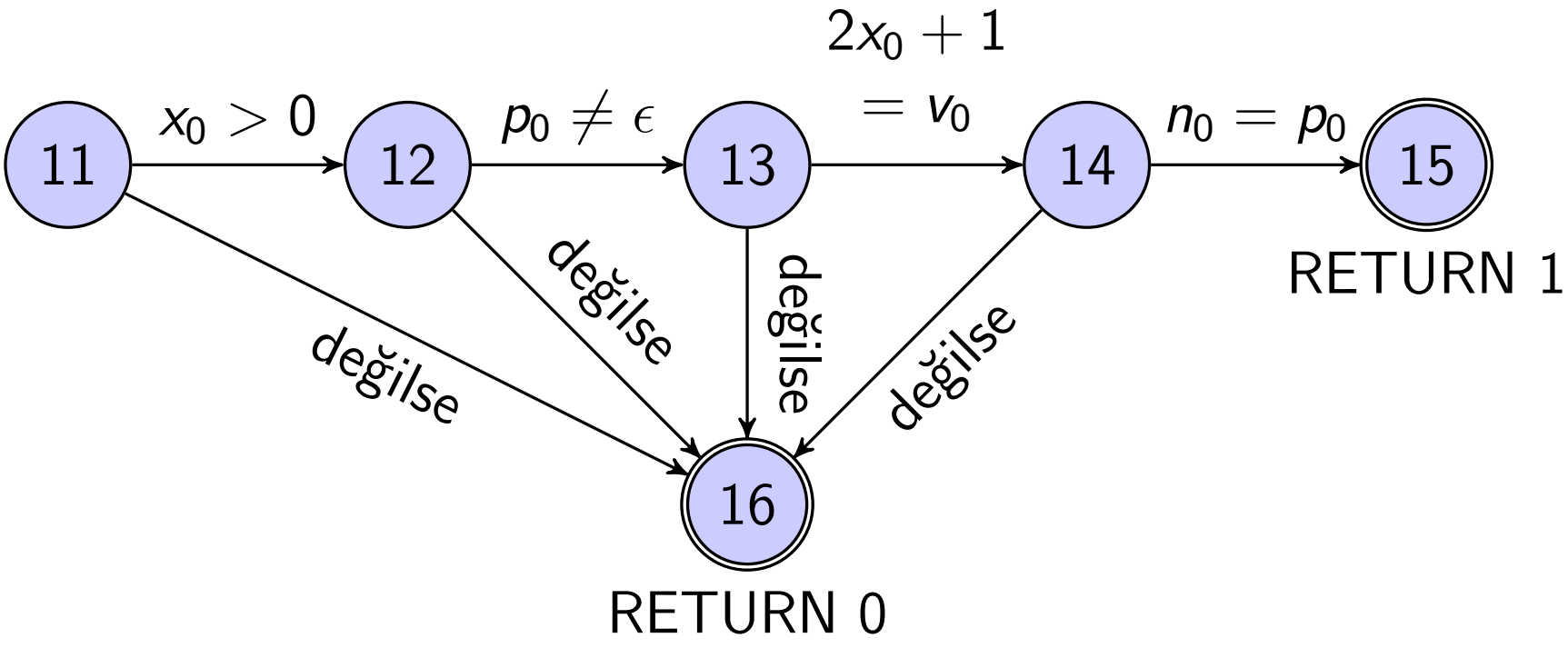
Büyüyen Kısmi Yol Kısıtları İle İcra

$$\pi_3 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, n_0 = p_0\}$$



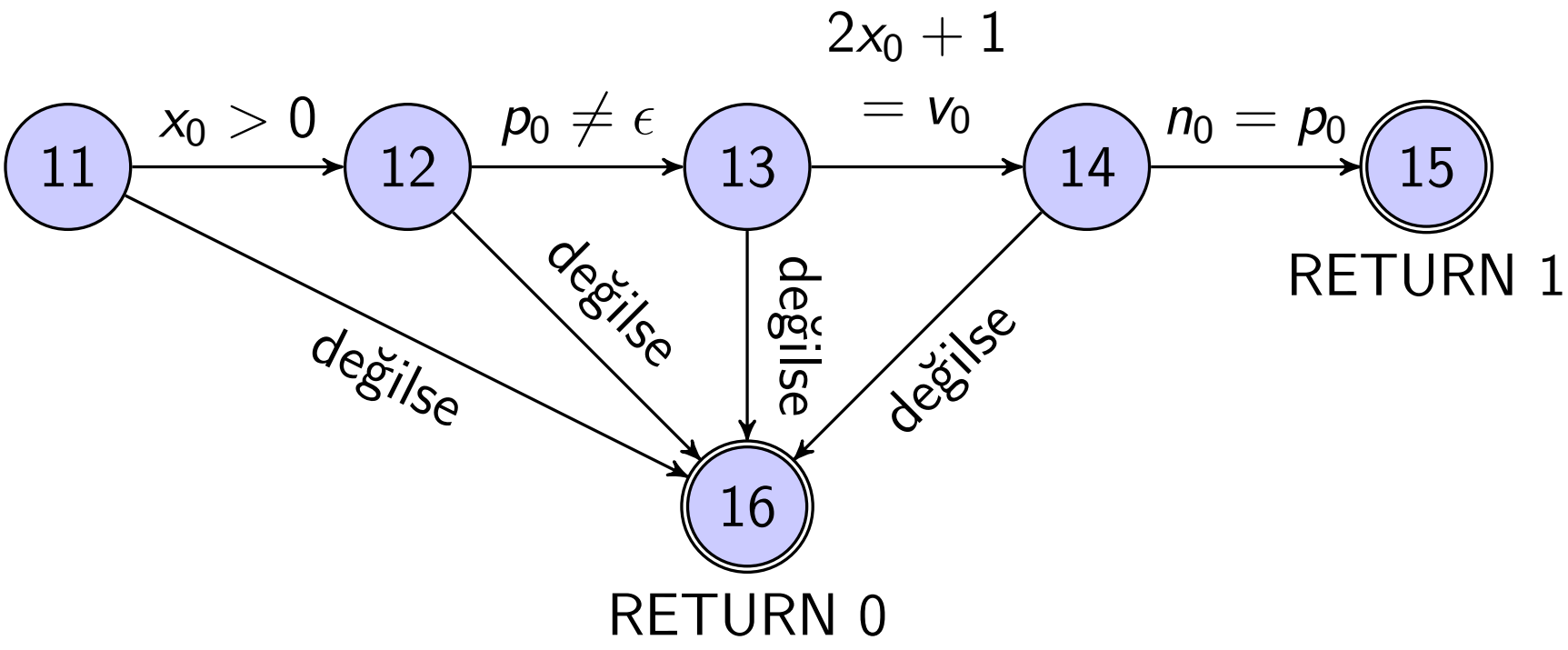
Büyüyen Kısmi Yol Kısıtları İle İcra

π_3 in sadece son koşulunu kısmi kısıt olarak al



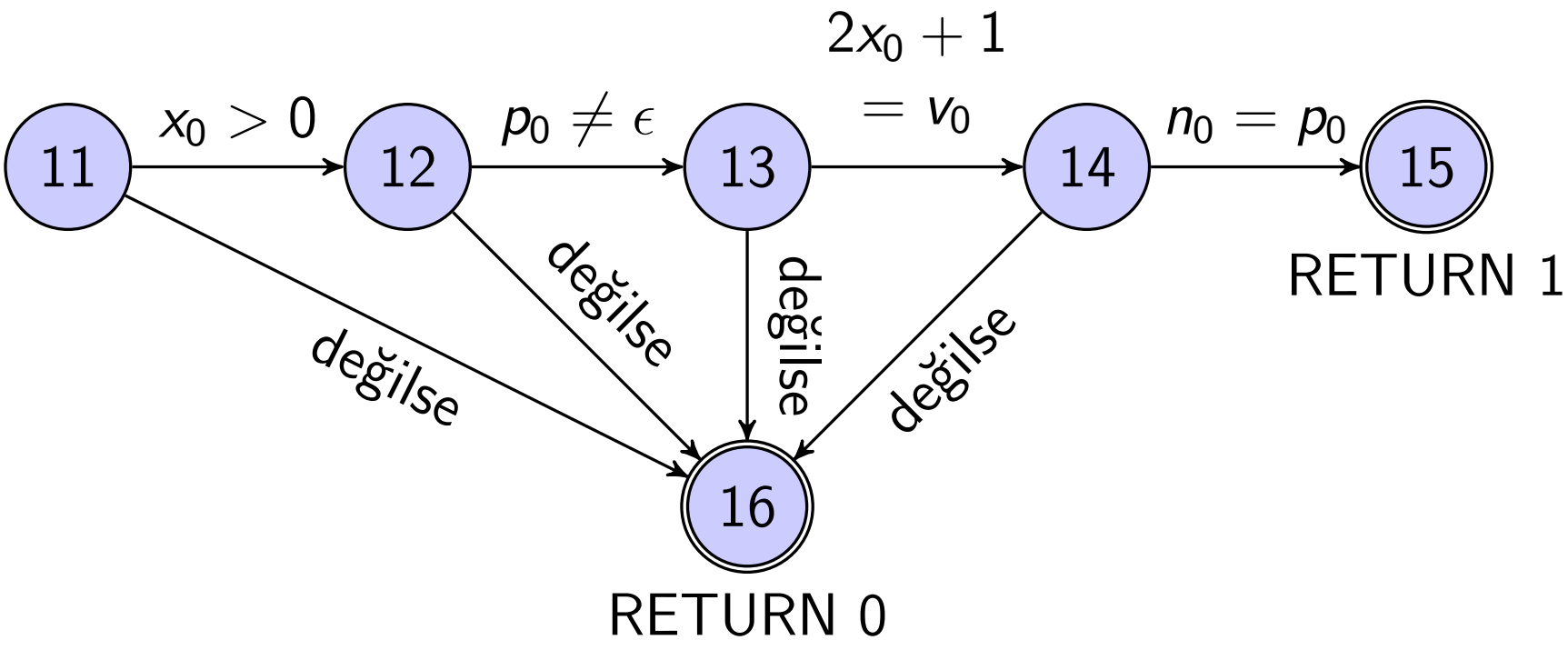
Büyüyen Kısmi Yol Kısıtları İle İcra

ÇÖZ($\phi = \{n_0 = p_0\}$)



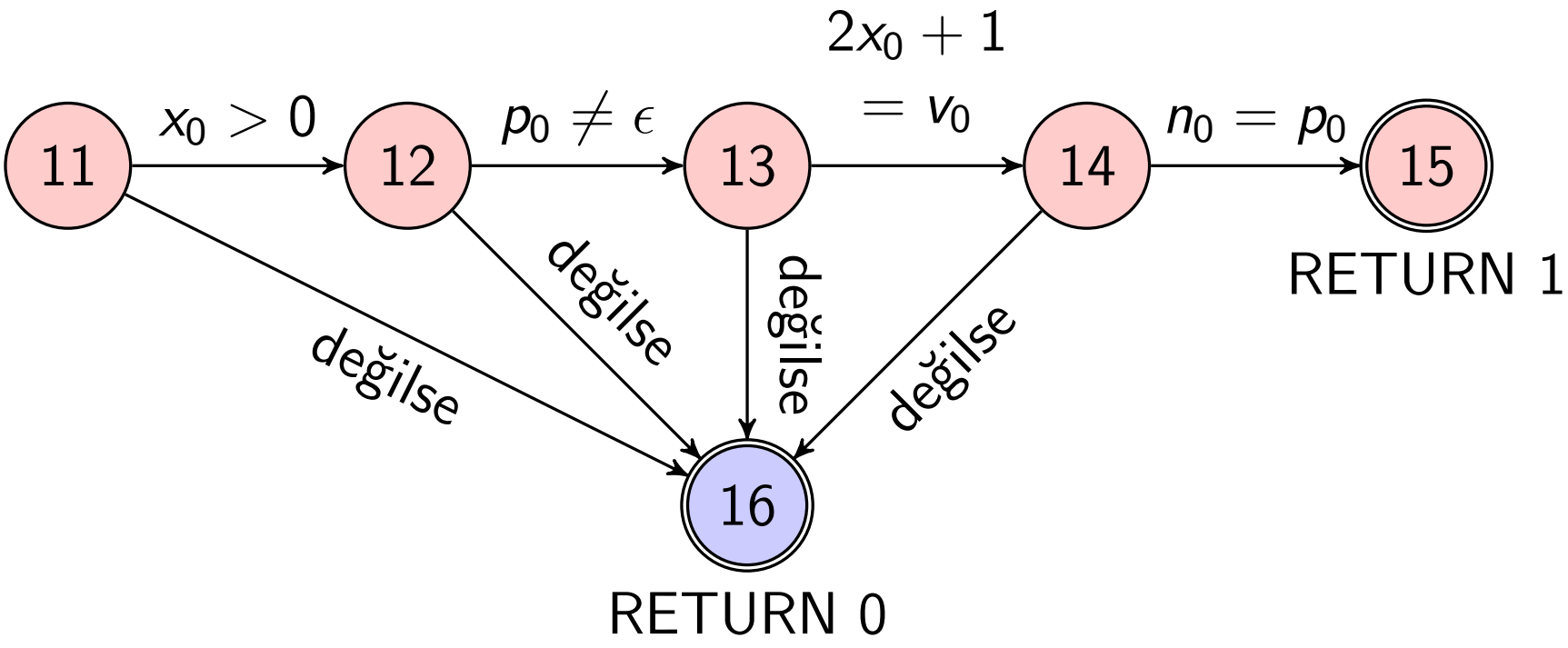
Büyüyen Kısmi Yol Kısıtları İle İcra

$= [x_0 = 7, p_0 = 9, v_0 = 15, n_0 = 9]$



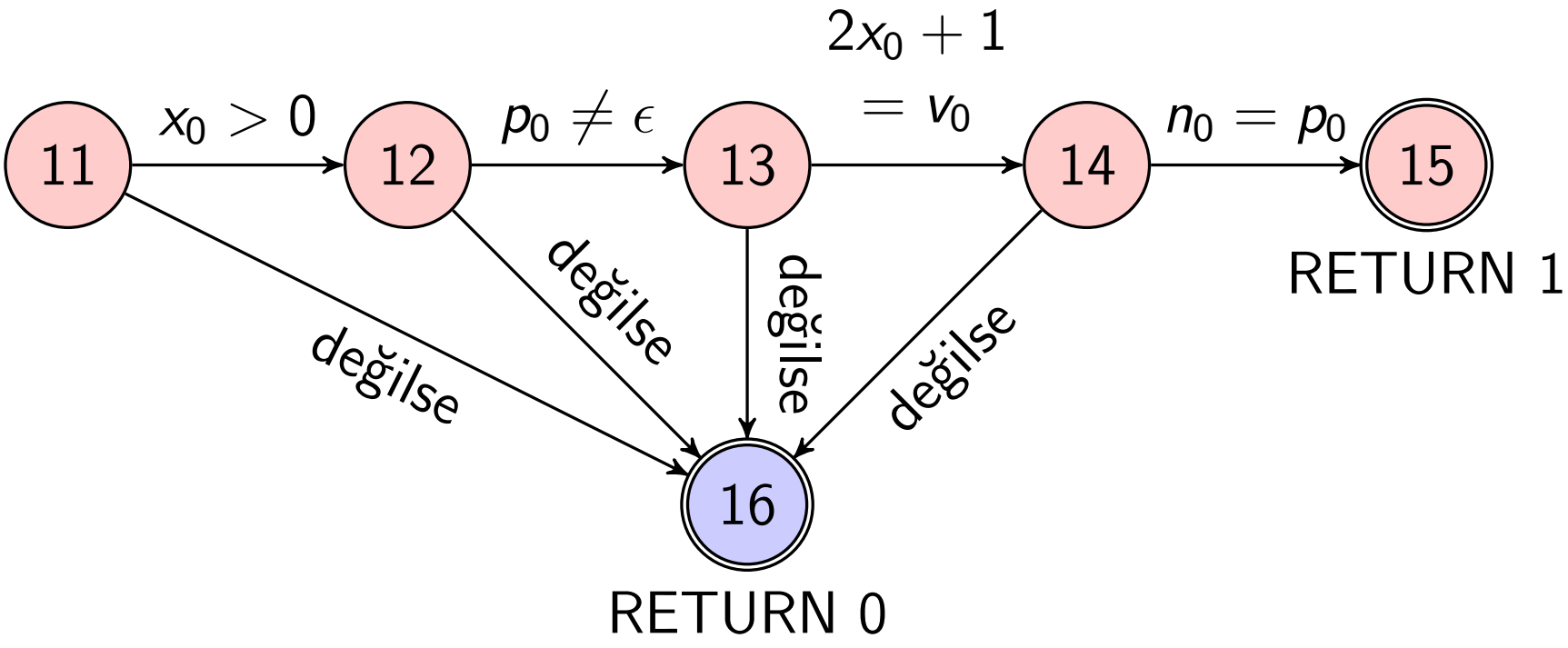
Büyüyen Kısmi Yol Kısıtları İle İcra

Programı çalıştır ve sembolik kısıtları topla



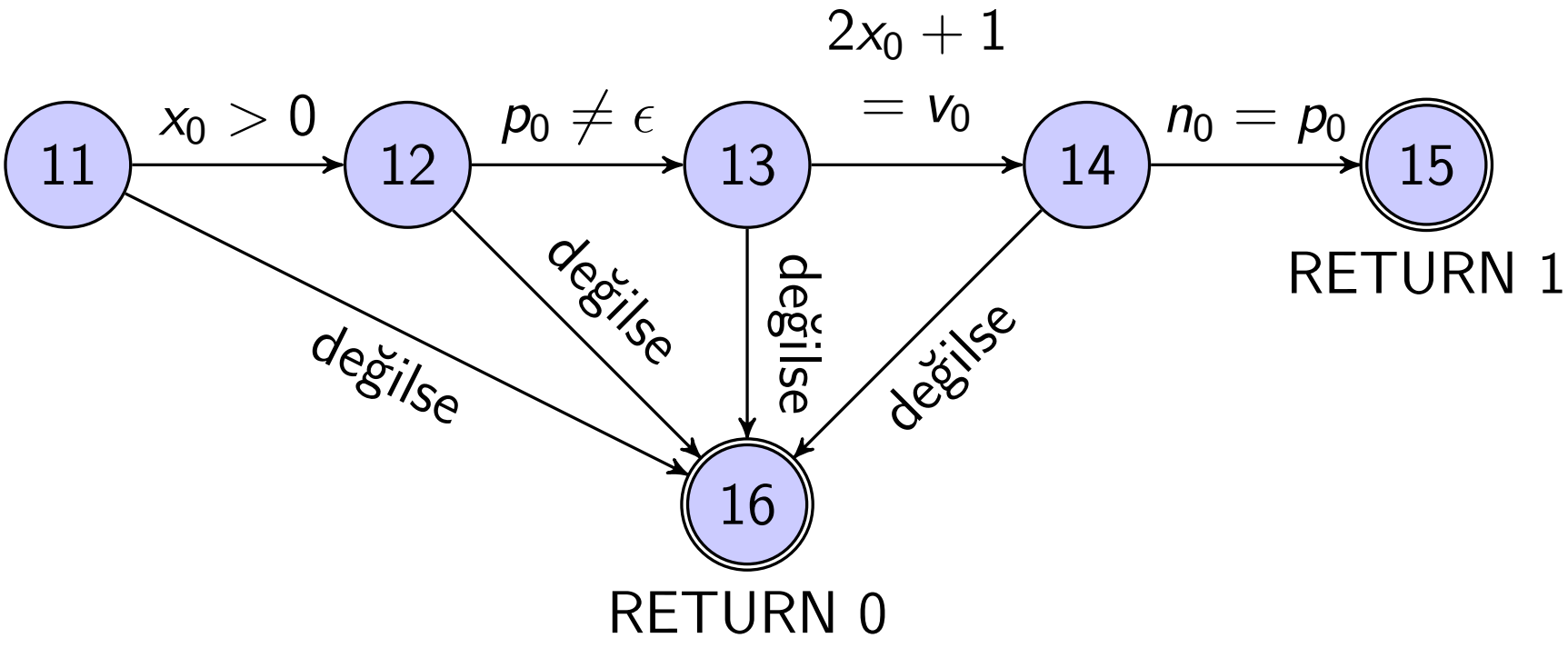
Büyüyen Kısmi Yol Kısıtları İle İcra

$$\pi_4 = \{x_0 > 0, p_0 \neq \epsilon, 2x_0 + 1 = v_0, p_0 = n_0\}$$



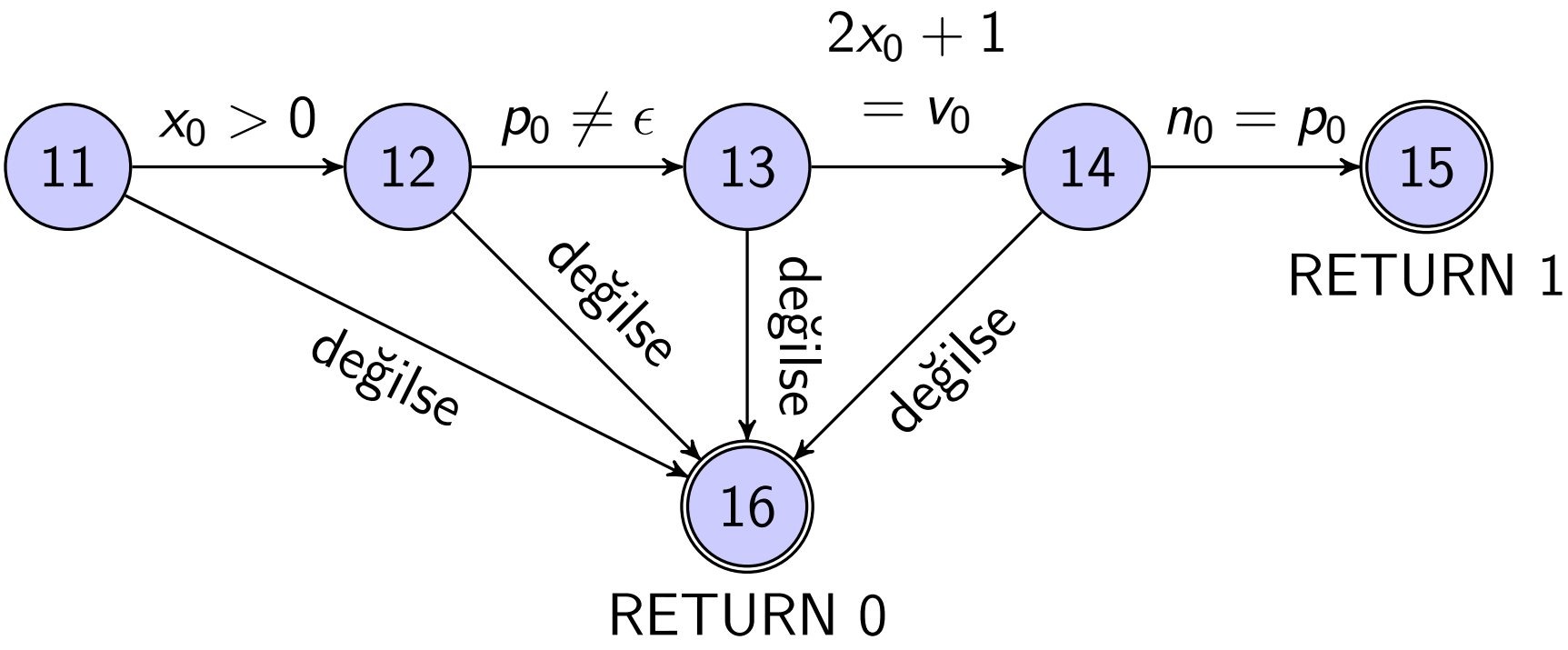
Büyüyen Kısmi Yol Kısıtları İle İcra

π_4 kısıtının **denenmemiş** son koşulunu ters çevir.



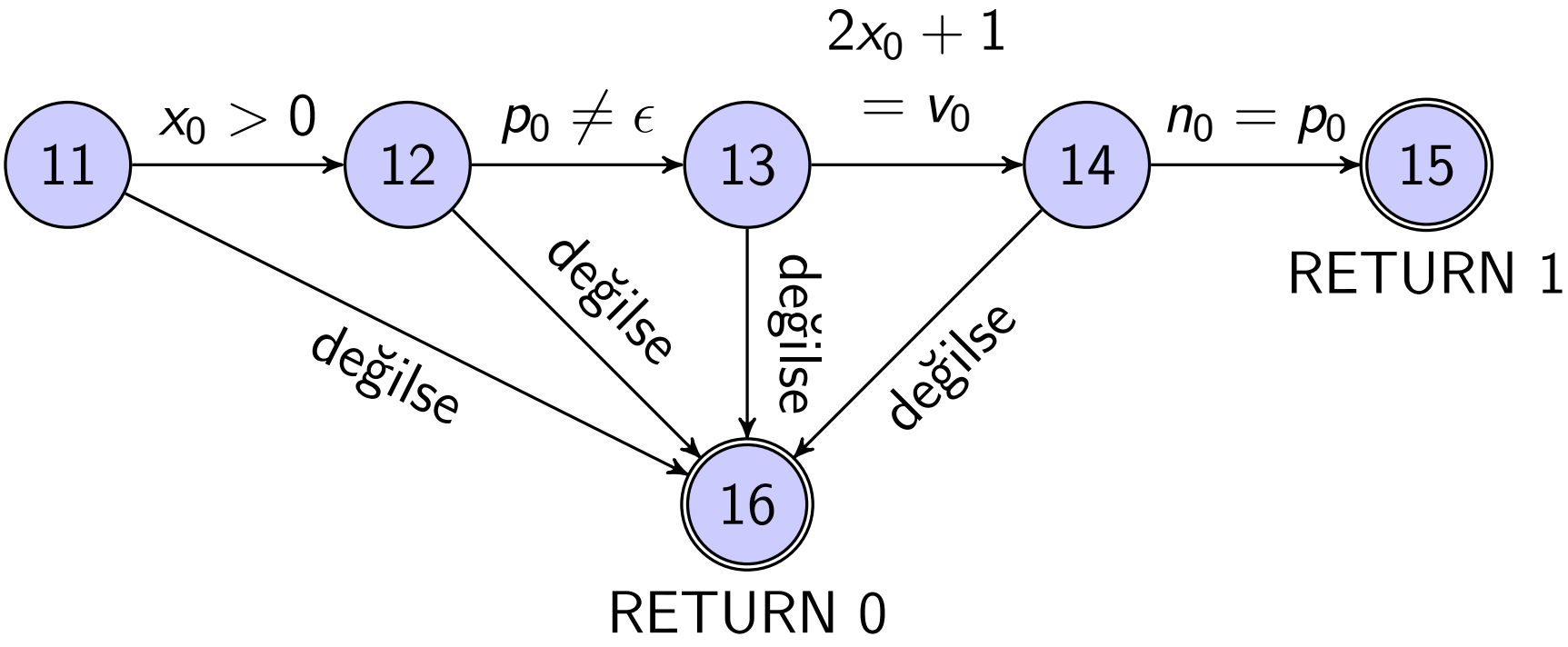
Büyüyen Kısmi Yol Kısıtları İle İcra

$\pi_5 = \{x_0 > 0, p_0 = \epsilon\}$



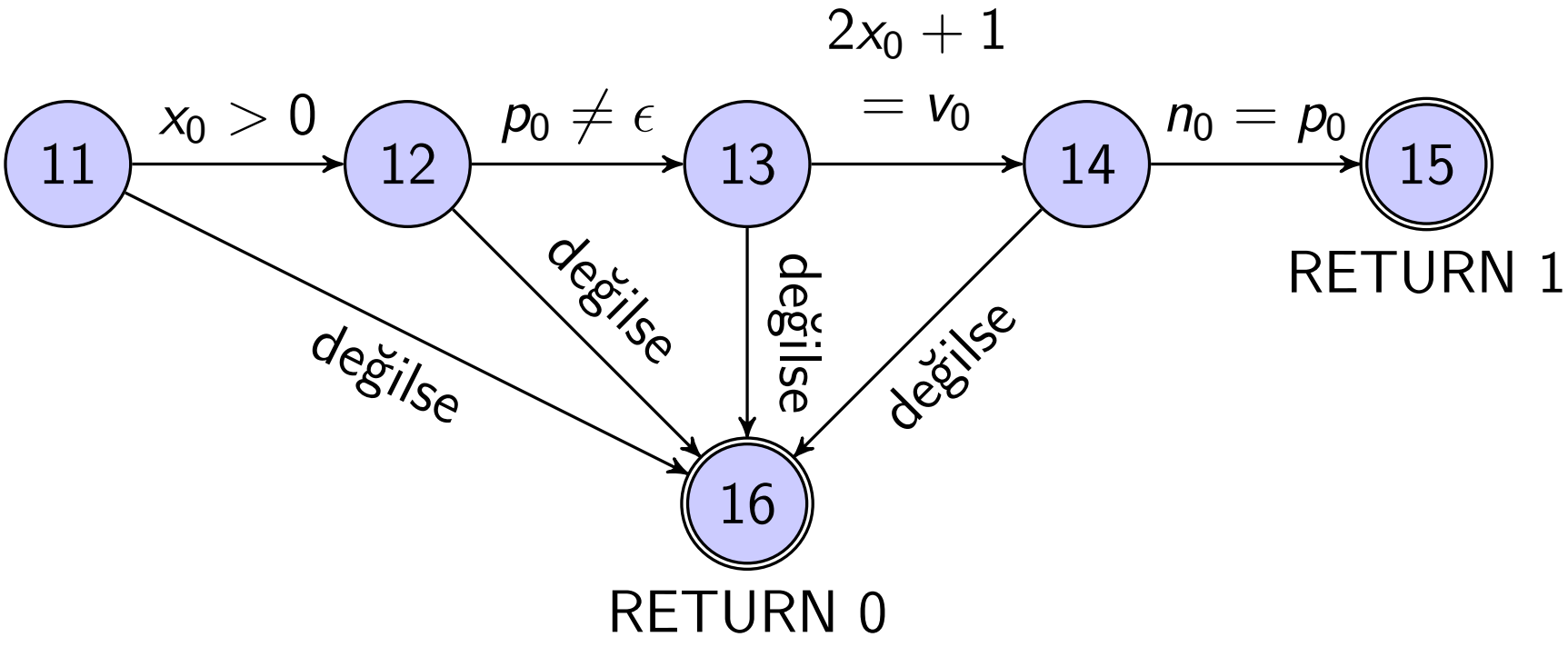
Büyüyen Kısmi Yol Kısıtları İle İcra

π_5 in sadece son koşulunu kısmi kısıt olarak al



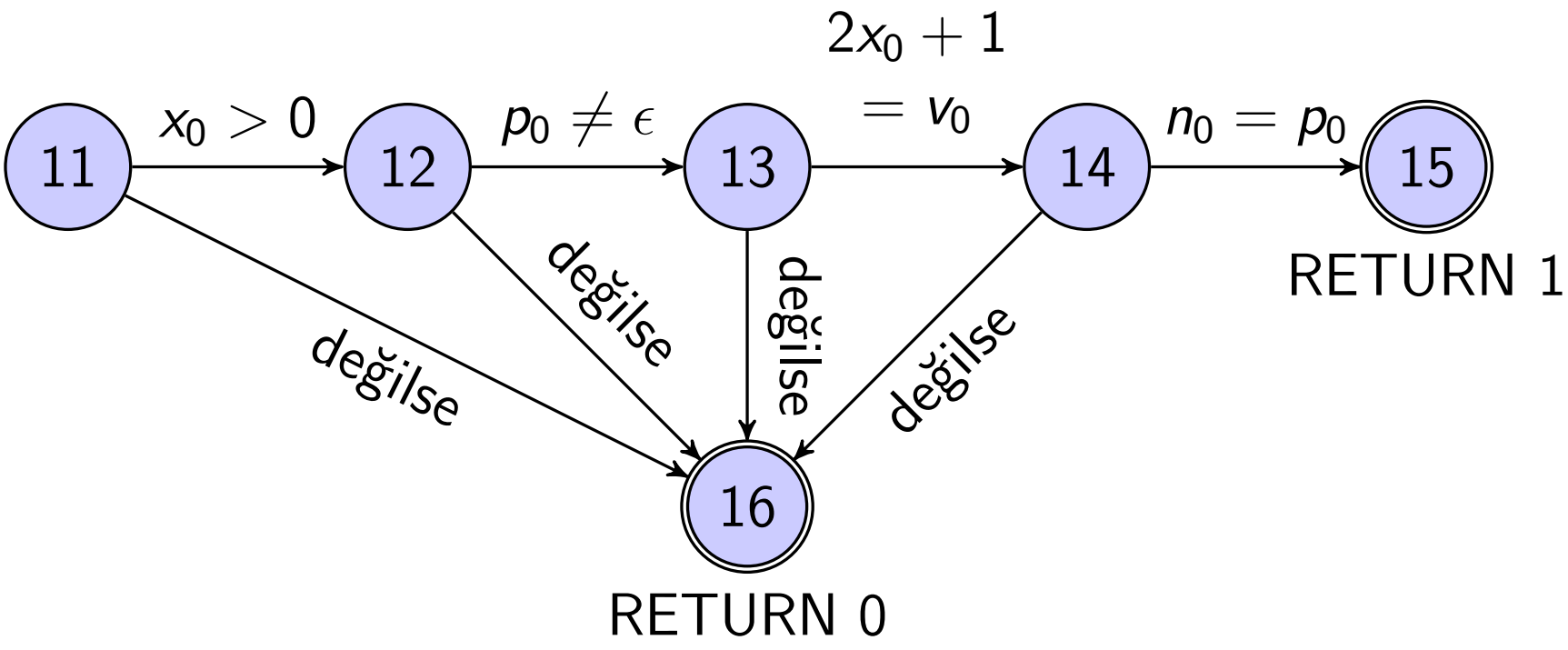
Büyüyen Kısmi Yol Kısıtları İle İcra

ÇÖZ($\phi = \{p_0 = \epsilon\}$)



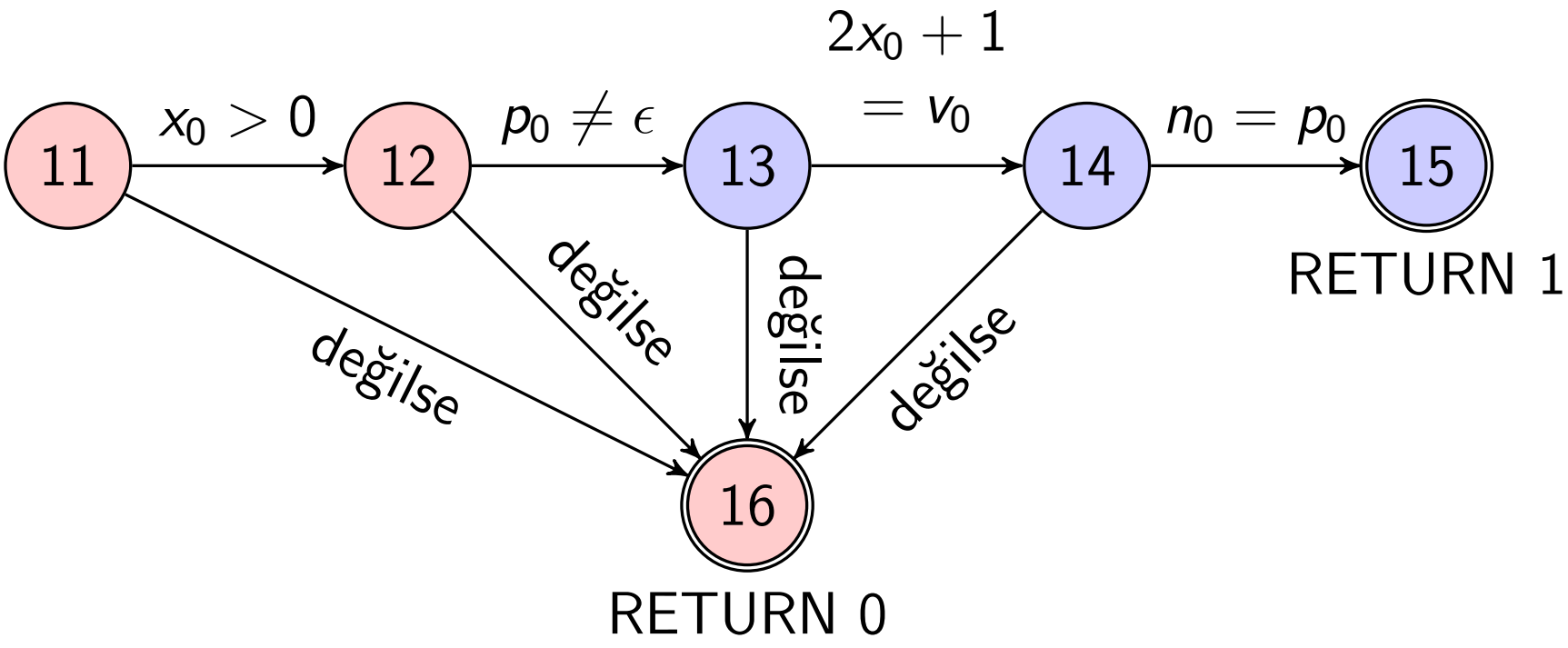
Büyüyen Kısmi Yol Kısıtları İle İcra

$= [x_0 = 7, p_0 = \epsilon, v_0 = 15, n_0 = 9]$



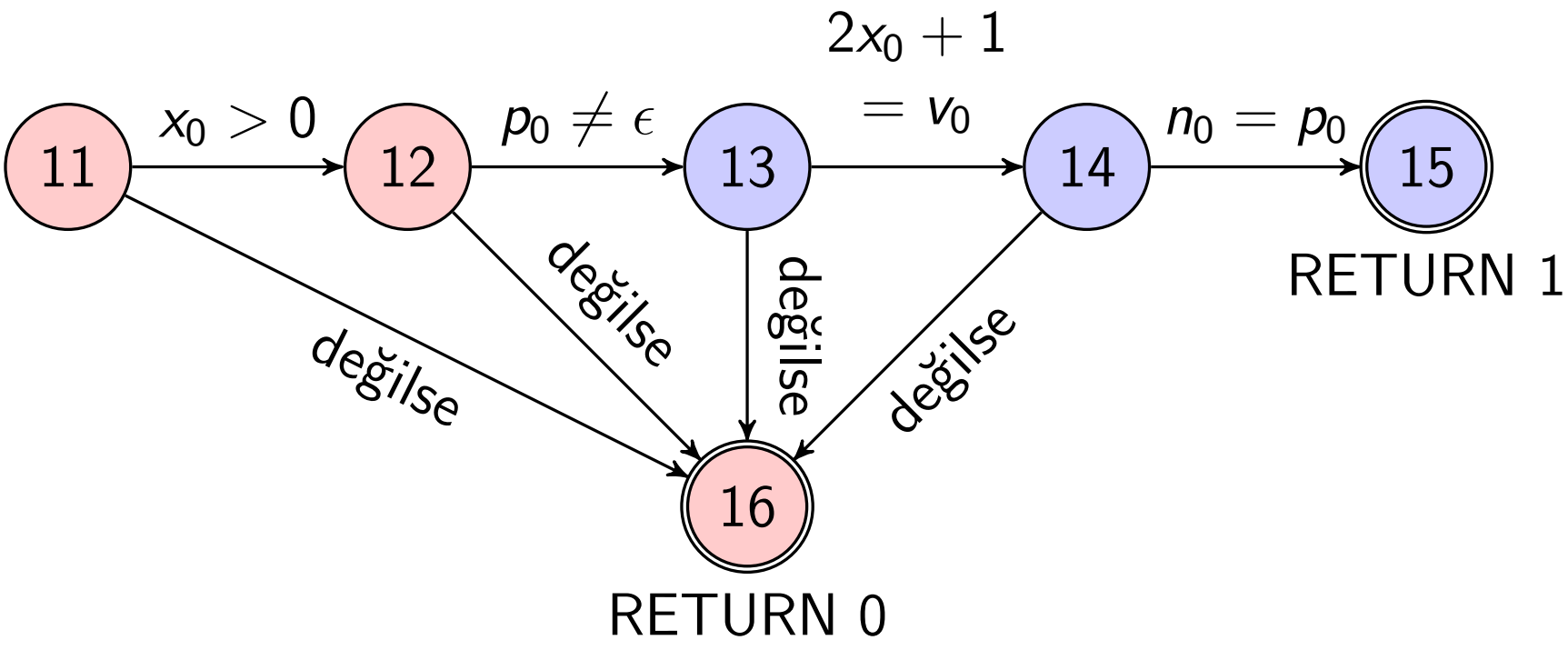
Büyüyen Kısmi Yol Kısıtları İle İcra

Programı çalıştır ve sembolik kısıtları topla



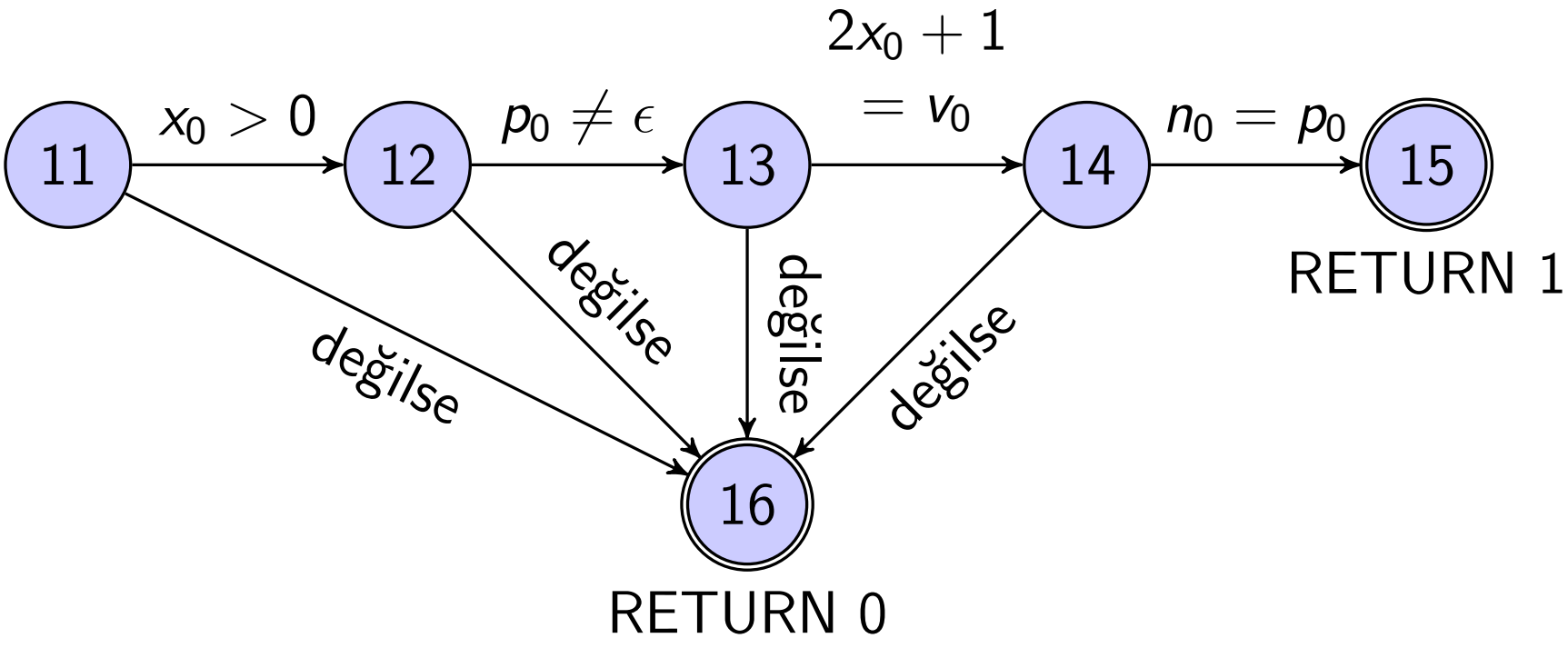
Büyüyen Kısmi Yol Kısıtları İle İcra

$\pi_6 = \{x_0 > 0, p_0 = \epsilon\}$



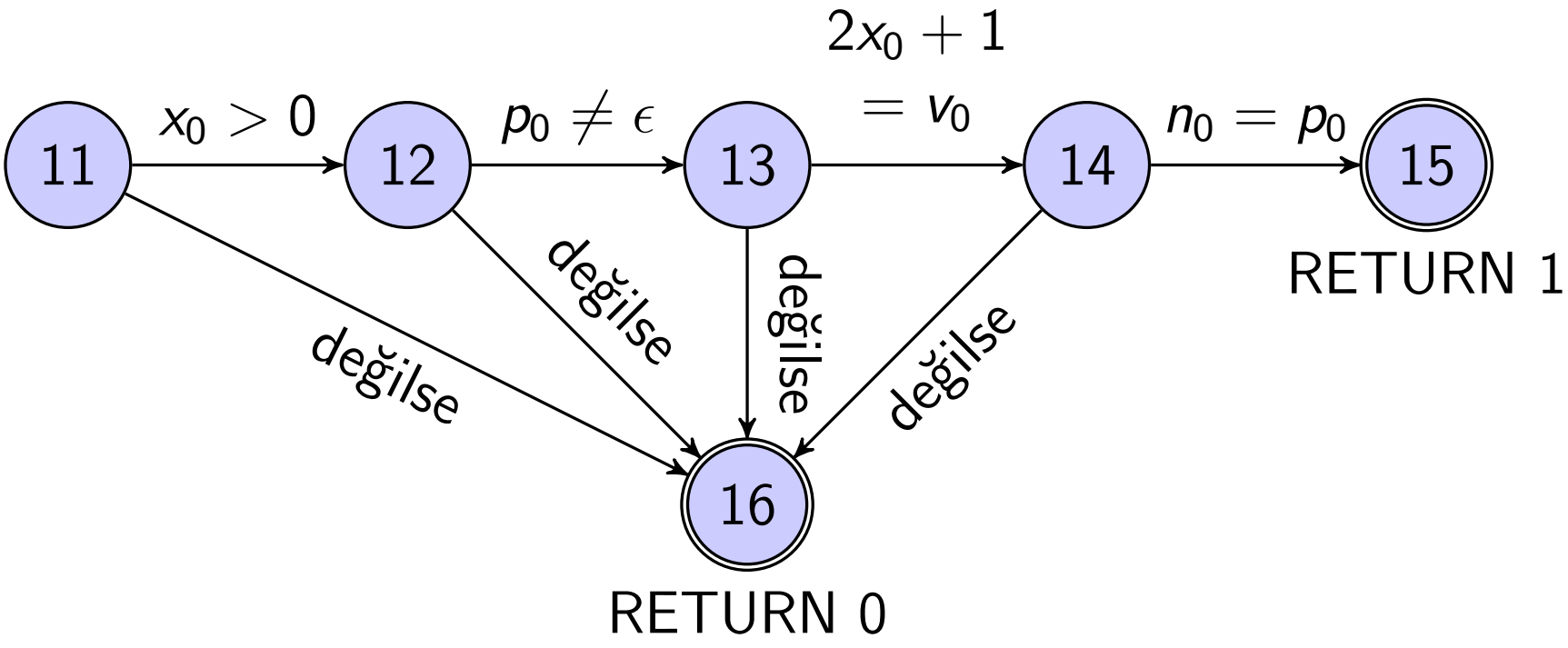
Büyüyen Kısmi Yol Kısıtları İle İcra

π_6 kısıtının **denenmemiş** son koşulunu ters çevir.



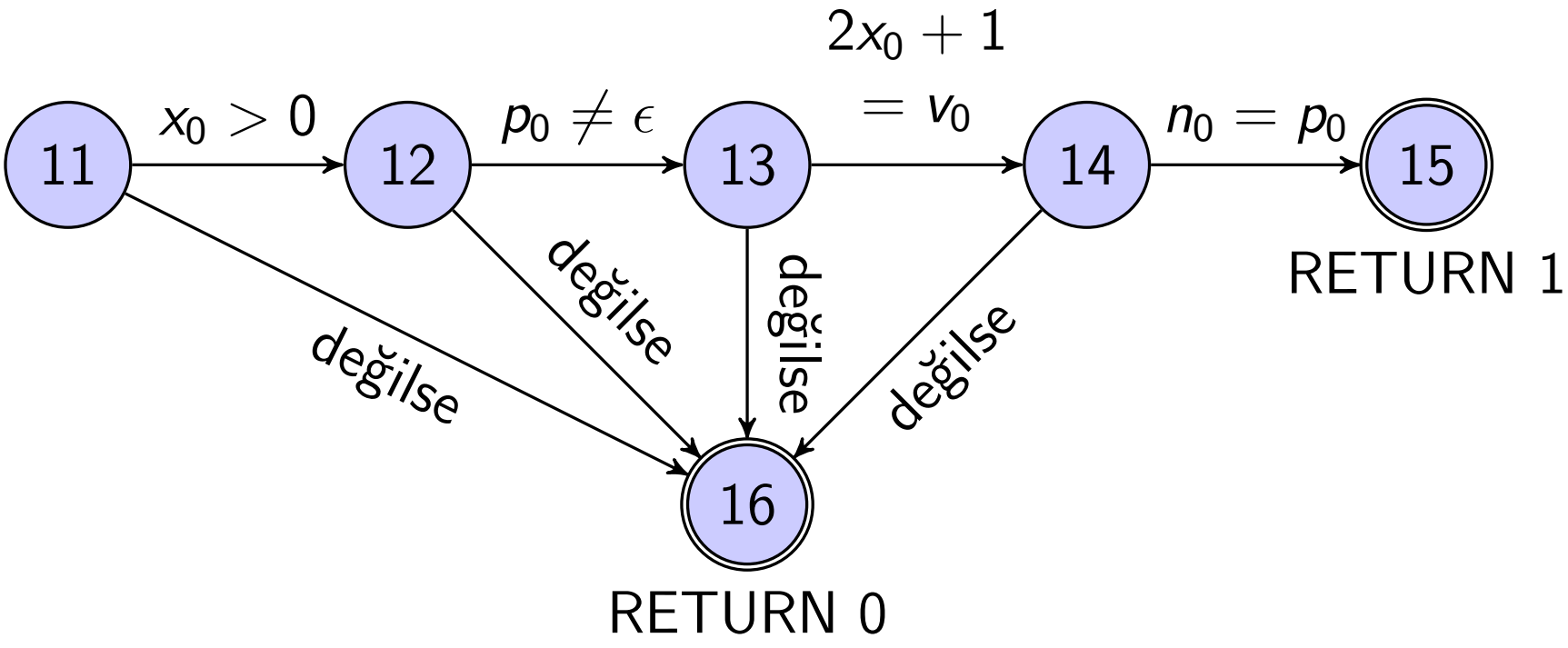
Büyüyen Kısmi Yol Kısıtları İle İcra

$\pi_7 = \{x_0 \leq 0\}$

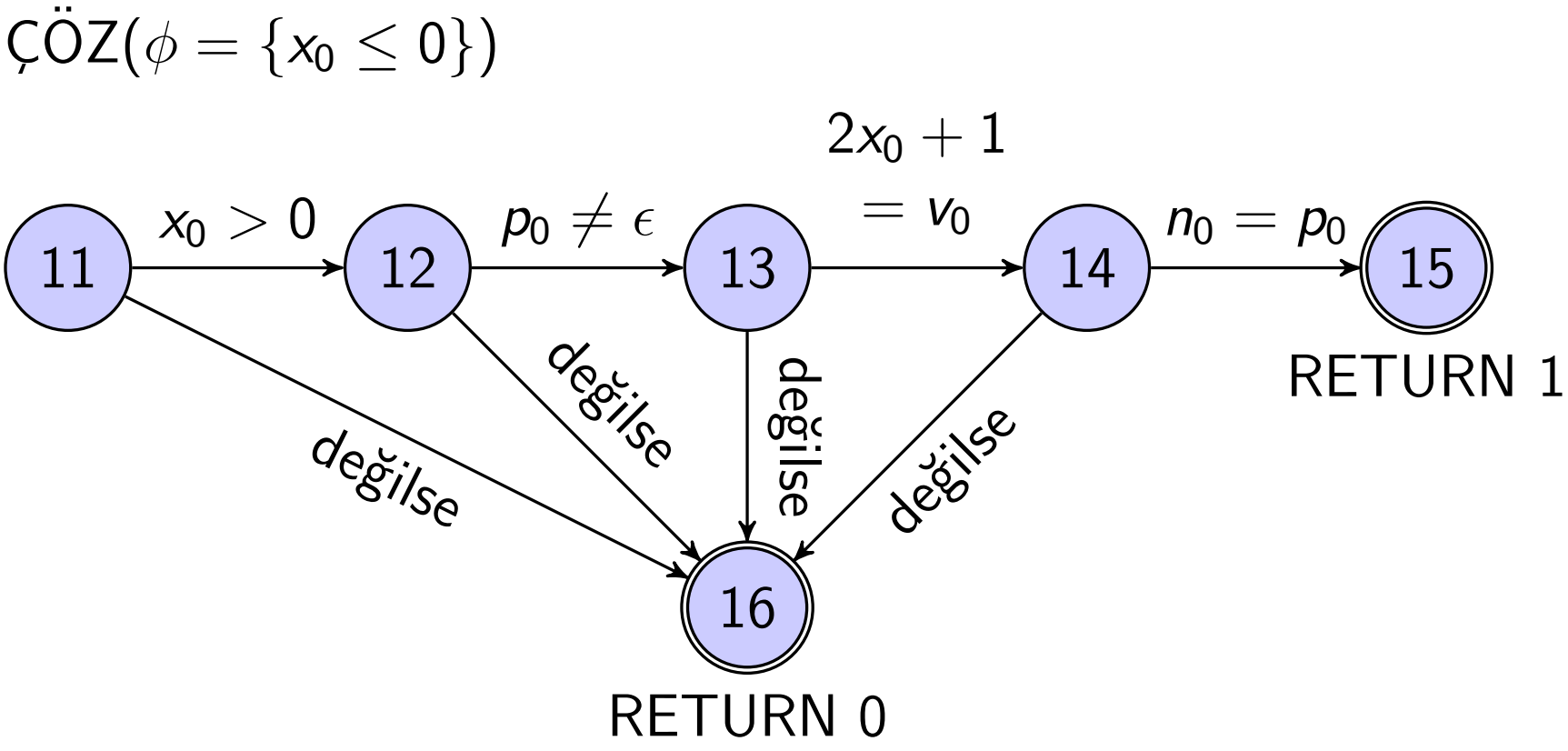


Büyüyen Kısmi Yol Kısıtları İle İcra

π_7 in sadece son koşulunu kısmi kısıt olarak al

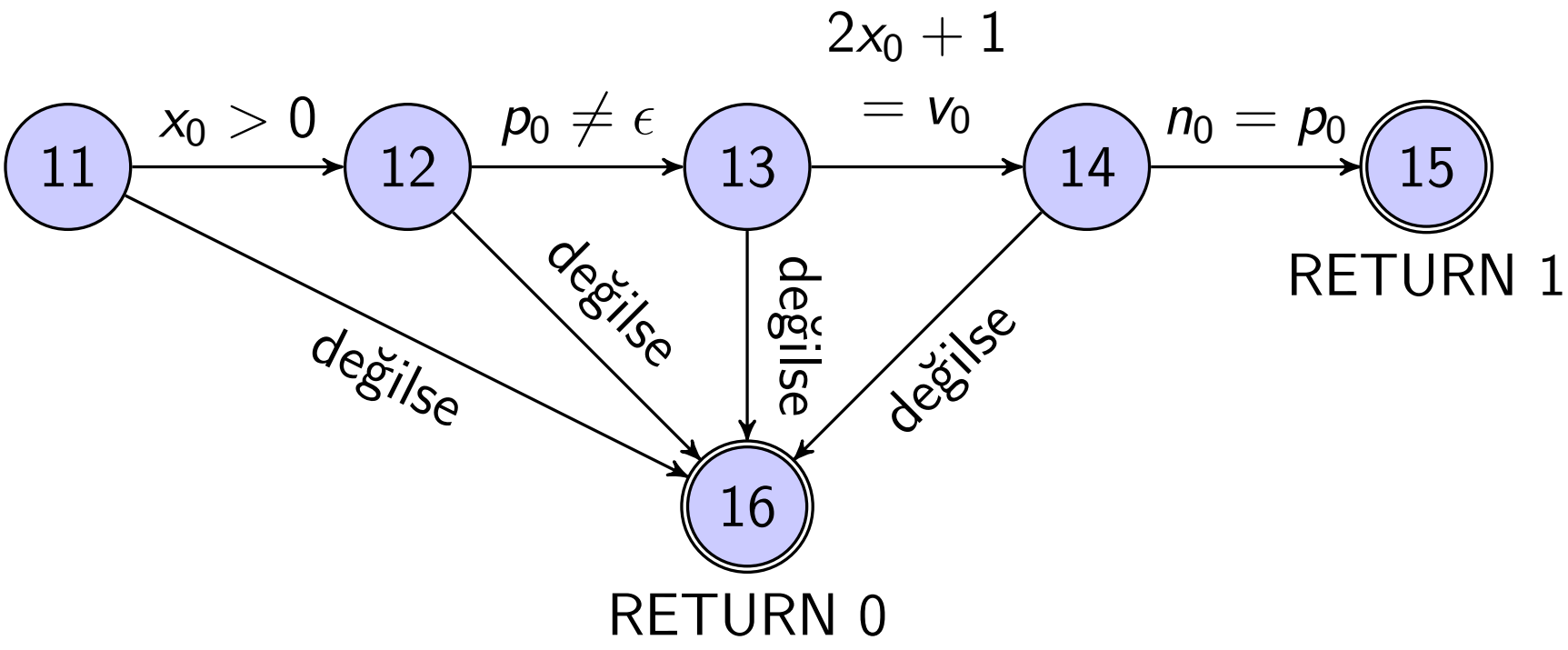


Büyüyen Kısmi Yol Kısıtları İle İcra



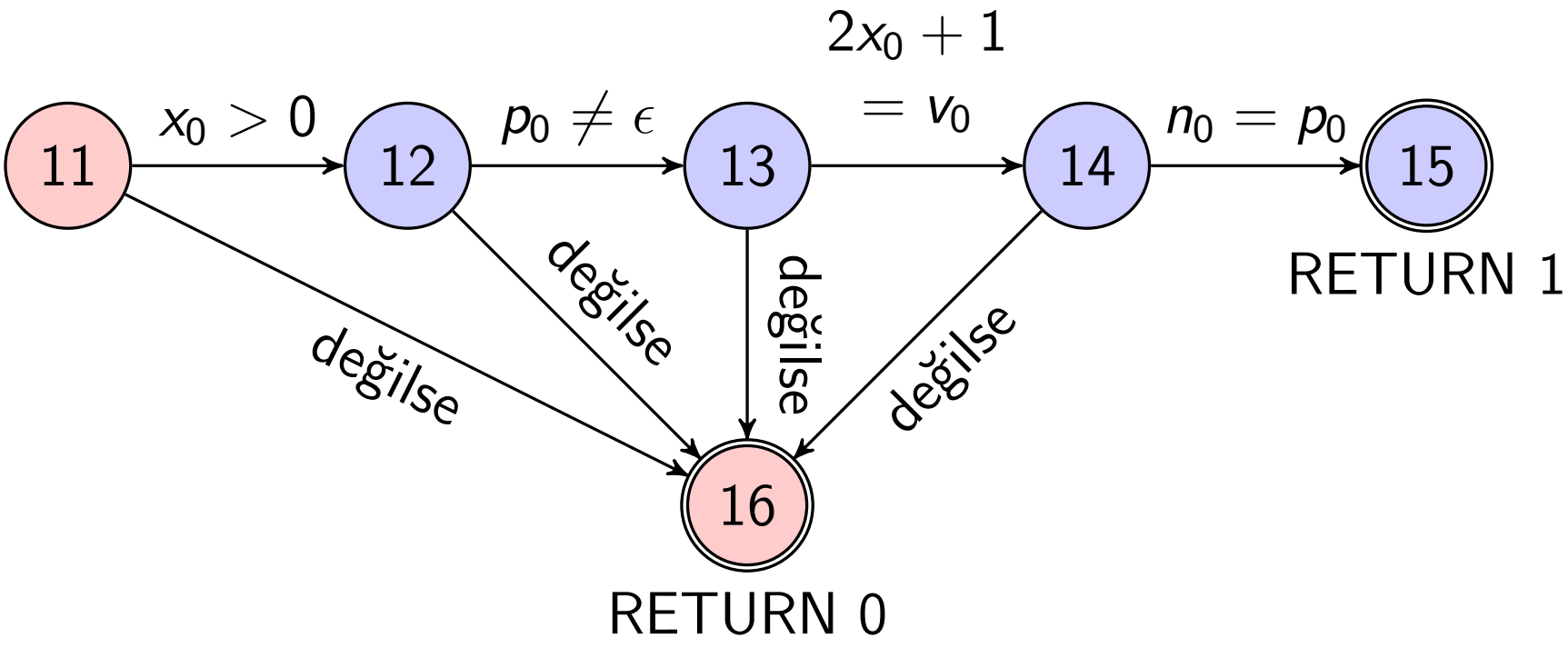
Büyüyen Kısmi Yol Kısıtları İle İcra

$= [x_0 = 0, p_0 = \epsilon, v_0 = 15, n_0 = 9]$

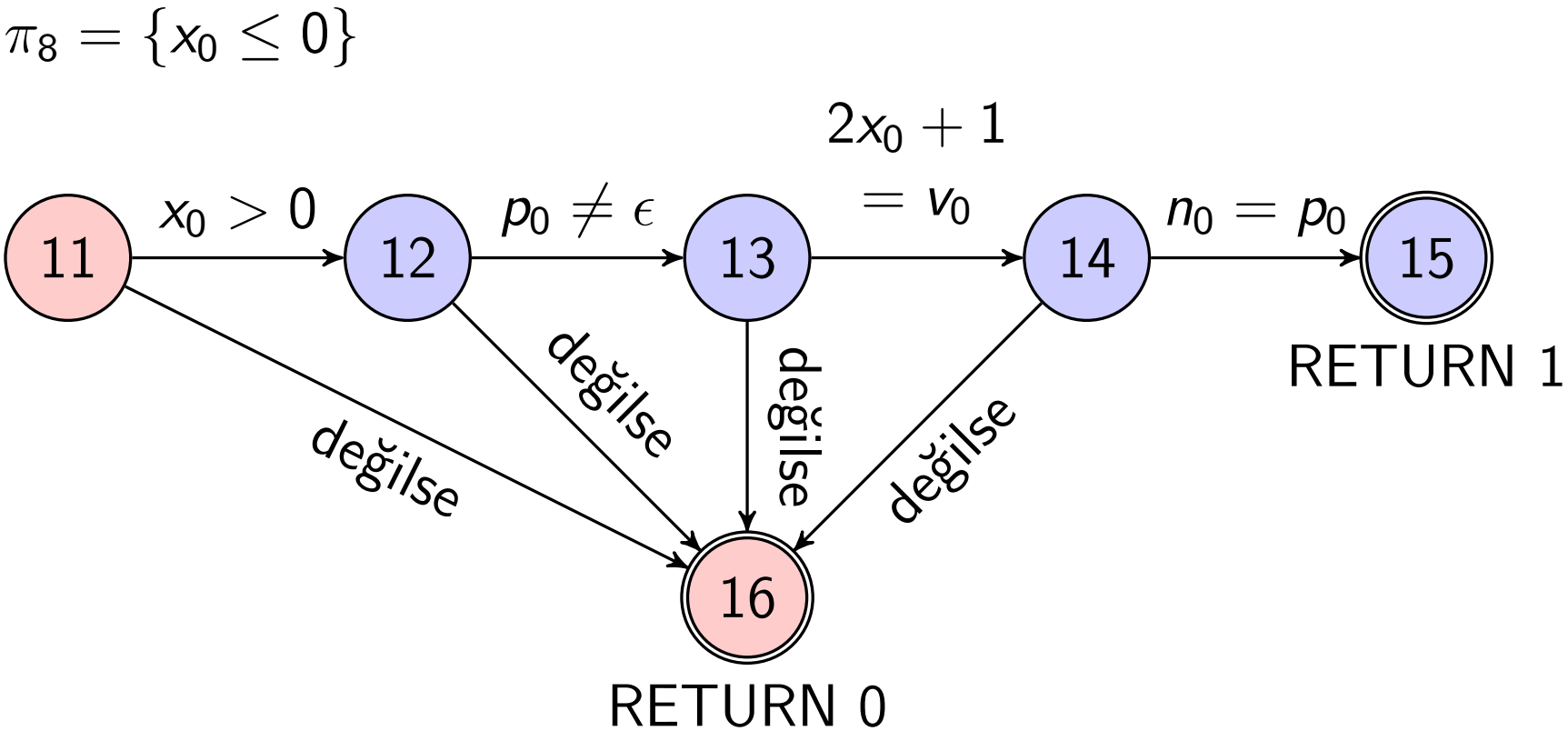


Büyüyen Kısmi Yol Kısıtları İle İcra

Programı çalıştır ve sembolik kısıtları topla



Büyüyen Kısmi Yol Kısıtları İle İcra



Büyüyen Kısmi Yol Kısıtları İle İcra

Sonuçlar

- Kısıt çözücü: 4 kere çağırıldı.
- Kısıt uzunluğu: Ortalama 1.
- İterasyon sayısı: 5.

Soru

Sonuçlar bu örneğe özel olarak iyi çıkmıştır. Pratikte bu algoritmanın performansı nedir?

Bellekli Kısmi Yol Kısıtları

Ön Çalışmalar Sonucunda,

- **Ortak kısmi kısıtlar** ve
- **Ortak test girdileri** gözlemlenmiştir.

Optimizasyon Fikri

- Tam yol kısıtlarıyla, o yolu çalıştıran girdilerin iki yönlü bir **haritası** çıkarılır.
- Kısmi kısıt $\rightarrow$ önceki girdiler sağlıyor $\rightarrow$ kısıt çözücü kullanma.
- Kısmi kısıt $\rightarrow$ hiç sağlayan girdi yok $\rightarrow$ girdi yarat $\rightarrow$ girdiler önceki bir tam yol kısıtını sağlıyor $\rightarrow$ programı hiç çalıştırma.

Ön Çalışma

Test Edilen Ufak Çaplı Programlar

- **Üçgen:** Kenar uzunlukları (a, b, c) verilen üçgeni sınıflandırma; `üçgen değil`, `çeşitkenar`, `ikizkenar`, `eşkenar`
- **Dörtgen:** Üç kenarı ve iki açısı verilen dörtgeni sınıflandırma; `yamuk`, `paralelkenar`, `eşkenar dörtgen`, `kare`, `hiçbiri`

Ön Çalışma

Deneye Katılan Test Yöntemleri

1. Derinlik öncelikli saf konkolik test
2. Kontrol-akış yönelimli konkolik test
3. Rastgele kısmi kısıtlar ile test (100 rastgele kısıt ile)
4. Belleksiz büyüyen kısmi yol kısıtlarıyla test
5. Bellekli büyüyen kısmi yol kısıtlarıyla test

Üçgen Programının Sonuçları

Yöntem No.	# Sorgu	Ort. Uzunluk	# Girdi	Kapsama
1	10	5.3	11	100%
2	19	4.789	19	100%
3	100	2.26	100	85%
4	26	2.769	27	100%
5	26	2.769	19	100%

Dörtgen Programının Sonuçları

Yöntem No.	# Sorgu	Ort. Uzunluk	# Girdi	Kapsama
1	10	5.5	11	100%
2	18	5.06	18	100%
3	100	1.77	100	65%
4	17	1.5	18	100%
5	17	1.5	18	100%

Yorum

- Sorgu sayısı artarken ortalama sorgu uzunluğunun kısalması.
- Kısıt çözücü üzerindeki yükün hafiflemesi.
- Daha büyük yazılımlarda bir performans artışı beklenmekte.

İleriki Çalışmalar

Daha İyi Kısmi Kısıtlar

- Kısıt çözücüler için birbirinden **bağımsız** koşulları çözmek kolay (Ör: $a = 95, b < c$).
- Başlangıç kısıtı olarak son koşul ile başlayan, birbirinden bağımsız bir koşullar kümesiyle başlamak.

Büyük Programlar

- Yöntemimizin **grep**, **vim** ve **replace** gibi pratikte kullanılan kodlar üzerinde denenmesi.

References I


Robert S. Boyer, Bernard Elspas, and Karl N. Levitt.
Select: A formal system for testing and debugging programs by symbolic execution.
In *Proceedings of the International Conference on Reliable Software*, 1975.


J. Burnim and K. Sen.
Heuristics for scalable dynamic test generation.
In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, 2008.

References II


Joe W. Duran and Simeon C. Ntafos.
An evaluation of random testing.
IEEE Trans. Softw. Eng., 10(4):438–444, 1984.


James C. King.
Symbolic execution and program testing.
Commun. ACM, 19(7):385–394, 1976.


Anil Kumar and Jerry St.Clair.
Cunit, http://cunit.sourceforge.net.

References III

Atif M. Memon and Myra B. Cohen.
Automated testing of gui applications: Models, tools, and controlling flakiness.
In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, 2013.

Xiao Qu and Brian Robinson.
A case study of concolic testing tools and their limitations.

In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, 2011.

References IV

Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs.
Jalangi: A selective record-replay and dynamic analysis framework for javascript.
In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, 2013.

Koushik Sen, Darko Marinov, and Gul Agha.
Cute: A concolic unit testing engine for c.
SIGSOFT Softw. Eng. Notes, 30(5):263–272, 2005.

References V


Nikolai Tillmann and Jonathan De Halleux.
Pex: White box test generation for .net.
In *Proceedings of the 2Nd International Conference on Tests and Proofs*, TAP'08, 2008.


D. T. Wang.
Properties of faults and criticalities of values under tests for combinational networks.
IEEE Trans. Comput., 24(7):746–750, 1975.