Neden-Sonuç Çizgelerinden Test Girişlerinin Oluşturulması

Deniz Kavzak[1,2], Tolga Ayav[1] ve Fevzi Belli[1]

[1] İzmir Yüksek Teknoloji Enstitüsü
Bilgisayar Mühendisliği Bölümü, 35430 Urla, İzmir

[2] Yaşar Üniversitesi
Yazılım Mühendisliği Bölümü, 35100 Bornova, İzmir

deniz.kavzak@yasar.edu.tr, tolgaayav@iyte.edu.tr, belli@upb.de

Özet. Neden-sonuç çizgeleri çok bilinen gereksinim tabanlı yazılım test yöntemlerinden biri olduğu halde Myers tarafından önerildiği 1979 yılından beri bu çizgelerden test girişleri üretilmesi konusunda yeterince kapsamlı çalışma yapılmamıştır. Bu bildiri, çizgelerin Boole ifadelerine dönüştürülmesini ve Boole ifadelerinin testi için önerilmiş olan MI, MAX-A ve CUTPNFP stratejilerini kullanarak test girişleri üretilmesini önermektedir. Mutasyon analizi kullanılarak bu yöntemlerin hata ortaya çıkarma başarımları ölçülmüş ve Myers'ın önerdiği orijinal test üretme yöntemiyle kıyaslanmıştır.
Anahtar Kelimeler: Yazılım testi, neden-sonuç çizgesi, modele dayalı test, mutasyon test analizi.

Abstract. Cause-effect graphing is a well-known requirement based testing technique. However, since it was introduced by Myers in 1979, there seems not to have been any sufficiently comprehensive studies to generate test cases from these graphs. This paper proposes to convert cause-effect graphs into Boolean expressions and find out the test cases using test input generation techniques for Boolean expressions, such as MI, MAX-A and CUTPNFP. Mutation analysis is used to compare the fault detection capabilities of these techniques and the results are also compared to the Myers' original test generation technique.
Keywords: Software testing, Cause-effect graph, model-based testing, mutation test analysis.

1 Giriş

Gereksinim tabanlı test yöntemleri, test senaryoları için koşullar ve bilgilerin gereksinimlerden çıkarsandığı yöntemlerdir. Gereksinim tabanlı testler, gereksinimlerin iyi belirlenmiş olduğu ve değişme ihtimalinin çok düşük olduğu durumlarda büyük avantaj sağlar. Gereksinimleri ifade etmek için kullanılan Boole ifadeler, modeller veya çizgeler aracılığı ile, kaynak kod henüz yazılmadan test senaryolarının oluşmasına imkan verir. Ancak test edilen program değişime açık bir sistem ise, her değişen gereksinim ile birlikte test senaryolarının baştan oluşturulması gerektiği için, gereksinim tabanlı bir test yöntemi kullanmak çok masraflı olacaktır. Bu nedenlerle, genellikle sağlık veya havacılık ile ilgili, gereksinimleri neredeyse değişmez olan kritik yazılımlar için kullanılır.

Gereksinim tabanlı test yöntemlerinden biri olan neden-sonuç çizgeleri (Cause-Effect Graph), birimlerin birbirleriyle bağımlılık ve ilişkilerini modelleyerek sistem gereksinimlerinin modellenmesi ve bu model üzerinden test üretilebilmesi için geliştirilmiştir [1, 2]. İlk olarak Myers [2], neden-sonuç çizgelerinden, model yollarından en az bir kere geçilmesini garanti ederek, giriş kombinasyonlarını kapsayacak bir karar tablosu yaratmak amacıyla sistematik bir yöntem tanıtmıştır. Bu yöntemle, giriş sayısı büyüdükçe çözülemeyecek boyutlara çıkan karar tablosu kombinasyonlarını (test girişlerinin sayısını) azaltmıştır. Myers'ın sunduğu algoritma tartışılmış [3], neden-sonuç çizgesinden karar tablosu oluşturmak için başka algoritmalar da sunulmuştur [4, 5].

Sistem gereksinimlerinin Boole ifadelerle gösterilebilmesinden yararlanarak MI, MAX-A [6], MUMCUT [7, 8], BOR, MC/DC [9], RC/DC [10] gibi bir çok test yöntemi sunulmuştur. Chen ve arkadaşları [11], Boole ifadelerden sundukları MUMCUT yöntemi ile test senaryosu üretilmesi için bir araç geliştirmiştir. Kaminski ve Ammann [12], MUMCUT'u geliştirerek Minimal-MUMCUT olarak adlandırdıkları, MUMCUT'un kriterlerini sağlayan, ancak aynı tip hataları daha az test girişi ile yakalayan bir yöntem geliştirmişlerdir. Sunulan bu yöntemlerin çoğunun ortak özelliği yöntemin uygulanabilmesi için Boole ifadenin Ayırıcı Normal Form'da (ANF), diğer bir deyişle çarpımların toplamı şeklinde ifadesinin gerekliliğidir. BOR yönteminin uygulanabilmesi için, ANF gereksinimine ek olarak Boole ifadenin tekil (singular) olması da gerekmektedir.

Singh ve arkadaşları [13], Boole ifadelerden test üretme yöntemlerini birlikte değerlendirmek ve karşılaştırmak için Elmendorf'un yöntemi [1], BOR, MC/DC ve RC/DC yöntemlerini incelemiştir. Boole türevleri kullanılarak neden-sonuç çizgelerinden formel olarak MC/DC test girişleri üretilmesi gösterilmiştir [14]. Fraser ve Gargantini [15], hata bulma işlemini mantıksal bir SAT problemi olarak formelleştirmiş, çözümünü MUMCUT yöntemi ile kıyaslamıştır. Sonraki çalışmalarında [16], bu çözüm genişletilerek Boole ifade genel formda iken çalışabilecek bir yöntem sunmuşlardır ve bu yöntemi MUMCUT ve MC/DC yöntemleri ile kıyaslamışlardır.

Sziray [17], neden-sonuç çizgelerinden test senaryoları üretmek için, karar ağacı kullanılan yeni bir algoritma geliştirmiştir. Vilkomir ve arkadaşları [18], t-li test üretme yönteminin Boole ifadeler üzerindeki etkinliğini mutasyon analizi kullanarak ölçmüş ve sonuçları rastgele test üretimi ile karşılaştırmıştır. Paul ve arkadaşları [19], daha önce tanıtılmış olan farklı MC/DC formlarını sistematik olarak incelemiş ve yeni bir MC/DC formu tanıtmışlar, bu formu eski MC/DC formları ile karşılaştırmışlardır. Chung [20], neden-sonuç çizgeleri için yeni hata sınıfları oluşturarak, Myers'ın yöntemi ve kombinatoryel test yöntemi ile ürettiği test kümelerinin, oluşturduğu bu hata sınıflarındaki hata yakalama başarılarını ölçmüştür. Chung [21], daha sonra, neden-sonuç çizgelerinden ikili test üretim yöntemi ile test senaryoları üreten otomatize edilmiş bir araç geliştirmiştir. Chung [22], başka bir bakış açısı ile, neden-sonuç çizgelerinden ikili test üretim yöntemi ile SAT çözücü kullanarak test senaryoları üretmiştir.

Sun ve arkadaşları [23], Boole ifadeler üzerinde tanımlanan test üretme yöntemlerinin bir kısmını (ONE, MIN, MUMCUT ve değişik formları,..) bir araya toplamış, rastgele ürettikleri Boole ifadeler üzerinde bu yöntemlerle ürettikleri test kümelerini çalıştırmış ve mutasyon analizi ile bu yöntemlerin hata yakalama başarılarını ölçmüşlerdir. Yöntemlerdeki ANF olma kısıtlarının ötesine geçmek için, genel formdaki ifadeler üzerindeki başarılarını da ölçmüşler, sonuç olarak

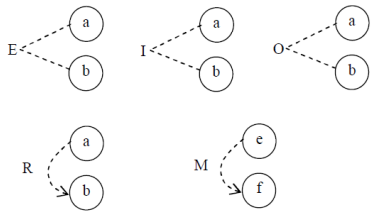
MUMCUT yönteminin daha başarılı olduğunu göstermiş ve MUMCUT yöntemine çeşitli eklentiler yapmışlardır.

Bu çalışmada neden-sonuç çizgesinden MI, MAX-A, CUTPNFP ve Myers yöntemleri kullanılarak test girişleri elde edilmiş, oluşan test girişleri mutasyon analizi ile değerlendirilmiş, yöntemlerin farklı hata sınıflarındaki başarıları karşılaştırılmıştır. Amaç, hangi yöntemin hangi hata sınıflarındaki hataları yakalama olasılığının daha yüksek olduğunu anlamaya çalışmaktır. Bölüm 2'de neden-sonuç çizgeleri ve özellikleri özetlenmiştir. Test giriş üretme yöntemlerinden, Bölüm 3.1'de MI, Bölüm 3.2'de MAX-A, CUTPNFP ve 3.3'te Myers açıklanmıştır. Bölüm 4'te hata tipleri ve mutasyon analizi sunulmaktadır. Bölüm 5'te oluşturulan neden-sonuç modeli ve araç mimarisi verilmiştir. Bölüm 6'da yapılan deneyler açıklanmış, değerlendirme ve sonuçlar paylaşılmıştır.

2 Neden-Sonuç Çizgesi

Neden-sonuç çizgesi, yazılım testlerinde kullanılan, nedenler kümesini sonuçlar kümesine eşleştiren, yönsüz bir çizgedir. Çizgenin elemanları sistemi oluşturan Boole ifadelerini görsel olarak temsil etmektedir. Neden düğümleri programa verilen, ikili sistemde değer alabilen, sistemin bir durumunu ya da bir etkinliğini tanımlayan girdiler, sistem gereksinim cümleleri, Boole ifadeleri olarak tanımlanabilmektedir. Sonuç düğümleri de, yine ikili sistemde değer alabilen, sistemin bir durumunu ya da bir sonuç etkisini belirten çıktıları olarak düşünülebilir.

Çizgenin genel yapısındaki düğümler soldan sağa: *neden düğümü* → *ara düğüm* → *sonuç düğümü* şeklinde tasarlanır. Çizgeler arasındaki ilişkiler, *ve, veya, değil* olmak üzere temel Boole ifadesi işlemleridir.



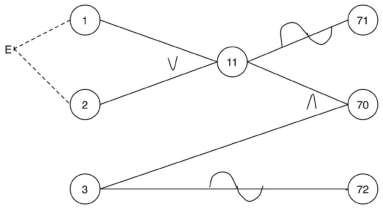
Şekil 1. Neden-sonuç çizgesinde tanımlanabilen koşullar

Bu ilişkilere ek olarak, *a, b* neden düğümleri, *e, f* sonuç düğümleri olmak üzere, Şekil 1'de gösterilen aşağıdaki koşullar tanımlanabilir:

- *Dışlayan veya (E)*: a ve b değişkenlerinden en fazla biri 1 değerini alabilir.
- *İçeren veya (I)*: a ve b değişkenlerinden en az biri 1 değerini almalıdır
- *Tekil (O)*: a ve b değişkenlerinden sadece ve sadece biri 1 değerini almalıdır.
- *Gerektirme (R)*: a değişkeninin 1 değerini alabilmesi için, b değişkeninin 1 olması gereklidir.
- *Maskeleme (M)*: e sonucu 1 değerini aldığında, f sonucu 0 değerini almak zorundadır.

Basit bir neden-sonuç çizgesi örneği olarak, sistem gereksinimlerinin şu şekilde tanımlandığını kabul edelim [2]: İki girdili sistemin ilk girdisi 'A' ya da 'B' harflerinden biri olmalı, ikinci girdi ise bir sayı olmalıdır. Girdiler doğruysa, dosya güncellemesi yapılır. İlk karakter yanlışsa, X12 hata mesajı, ikinci karakter sayı değilse X13 hata mesajı verilir. Bu gereksinimlerden çıkarılan neden ve sonuç düğümleri aşağıda, oluşan neden-sonuç çizgesi ise Şekil 2'de verilmiştir.

Nedenler:	Sonuçlar:
1- ilk girdi 'A'	70- dosya güncellemesi yapıldı
2- ilk girdi 'B'	71- X12 mesajı verildi
3- ikinci girdi bir sayı	72- X13 mesajı verildi



Şekil 2. Örnek neden-sonuç çizgesi

3 Test Girişi Üretme Yöntemleri

E'nin ANF olarak verilen, n terimli, m değişkenli bir Boole ifadesi olduğunu varsayalım:

$$E = e_1 + e_2 + ... + e_n \tag{1}$$

Her $e_i \quad (1 \leq i \leq n)$, l_j sayıda değişkenden oluşan bir terim olsun. Bu tanıma dayanarak takip eden bölümlerde MI, MAX-A ve CUTPNFP test üretme yöntemleri sunulacaktır.

3.1 MI

Anlamlı etki yöntemi (Meaningful Impact (MI)), Weyuker ve arkadaşları [6] tarafından TCAS-2 sisteminin Boole ifadeleri üzerinde çalışılarak tanımlanmıştır. ANF olarak verilen Boole ifadeleri üzerinde, özgün doğruluk noktaları (UTP) ve yakın yanlışlık noktaları (NFP) mantığı ile, eksik değişken hatası (EDH), değişken olumsuzlama hatası (DOH), değişken referans hatası (DRH), işlem referans hatası (İRH), ilişkisel kaydırma hatası, ifade olumsuzlama hatası (İOH) gibi hataları yakalayabilir. Ancak özgün doğruluk noktaları rastgele seçildiği için her zaman bu hataları ortaya çıkaramayabilir. Hatalı Boole ifadesindeki hatalı bileşen, orijinal Boole ifadesindeki başka bir bileşenin özgün doğruluk noktası veya yakın yanlışlık noktası ile çakışıyor olabilir.

MI algoritması aşağıdaki şekilde tanımlanmıştır [24]:

1. Her terim $e_i,\ 1 \leq i \leq n$ için, e_i'yi doğru yapan Te_i kümeleri oluşturulur.
2. Her $i \neq j,\ TSe_i \cap TSe_j = \emptyset$ için $TSe_i = Te_i - \bigcup_{j=1,i\neq j}^{n} Te_j$.
3. Her $TSe_i,\ 1 \leq i \leq n$ kümesinden birer eleman seçerek S_E^t oluşturulur.
4. $e_i^j,\ 1 \leq i \leq n$ ve $1 \leq j \leq l_j$ için e_i teriminin j.inci değişkeninin değilinin alınmasıyla oluşan terimler oluşturulur. e_i^j'yi doğru yapan Fe_i^j kümeleri oluşturulur.
5. $FSe_i^j = Fe_i^j - \bigcup_{k=1}^{n} Te_k$.
6. Her FSe_i^j kümesini en az bir kere örten ve minimal olan S_E^f oluşturulur.
7. E için gerekli toplam küme $S_E = S_E^t \cup S_E^f$

İlk olarak Boole ifadenin (E) her terimi (e_i) için, o terimi doğru yapan test giriş kümeleri (Te_i) oluşturulur. Bu oluşturulan kümelerden, her bir terim için, birbirlerinden ayrışık (kesişimleri boş küme) olacak şekilde alt kümeler (TSe_i) seçilir. Bu oluşan alt kümelerden, rastgele birer eleman seçilerek doğrular kümesi (S_E^t) oluşturulur. Daha sonra her terim (e_i) için oluşturulan olumsuz terimi (e_i^j'yi) doğru yapan test giriş kümeleri (Fe_i^j) oluşturulur. Bu kümelerden, ilk adımda oluşturulan doğru kümeler ile (Te_i) ortak olanlar varsa, bu ortak kümeler çıkarılarak oluşturulan alt kümeler (FSe_i^j) elde edilerek doğru ve yanlış test giriş kümelerinin hedeflediği durumların birbirleriyle karışması önlenir. Oluşturulan bu alt kümelerin hepsini içeren bir yanlışlar kümesi (S_E^f) oluşturulur. MI yöntemi için gerekli olan küme de bu iki kümenin birleştirilmesi ile oluşmaktadır.

3.2 MAX-A ve CUTPNFP

Weyuker ve arkadaşları [6], UTP ve NFP kümelerinin herhangi bir koşul olmadan birleştirilmesinden oluşan MAX-A yöntemini tanıtmışlardır. Chen ve arkadaşları [7, 8], üç farklı yöntemin (CUTPNFP, MUTP ve MNFP) birleşmesiyle oluşan MUMCUT yöntemini tanıtmışlardır. Bu üç yöntem tekrarsız (irredundant) ANF olarak verilen Boole ifadeler üzerinden gerçekleştirilmiştir. Boole ifadelerinin tekrarsız ANF olarak kısıtlanmasının sebebi test girişleri sayısını düşürmek ve karmaşıklığı azaltmaktır, performans ve hassasiyet derecesi arasındaki dengede bu seçim yapılmıştır. Bu çalışmada kanonik (her terim bütün değişkenleri bir kere içeren) ANF kullanıldığı için, MUMCUT'u oluşturan parçalardan CUTPNFP yöntemi incelenmiştir.

Tanım 1 *Özgün Doğruluk Noktası (UTP):* UTP_i, (1) *Boole ifadesinde* e_i *terimini doğru yapan, diğer tüm terimleri yanlış yapan test girişleri kümesidir. Boole ifadesi için tüm özgün doğruluk noktalar kümesi:* $UTP(S) = \bigcup_i UTP_i(S)$.

Tanım 2 *Yakın Yanlışlık Noktası (NFP):* $NFP_{i,\bar{j}}$, (1) *Boole ifadesinde* $e_{i,\bar{j}}$ *terimini (*e_i *teriminin* j*. değişkeninin değilinin alınmasıyla oluşan terim) doğru yapan, diğer tüm terimleri yanlış yapan test girişleri kümesidir. Boole ifadesinin* e_i *terimi için tüm yakın yanlışlık noktalar kümesi:* $NFP_i(S) = \bigcup_j NFP_{i,\bar{j}}(S)$. *Boole ifadesi için tüm yakın yanlışlık noktalar kümesi:* $NFP(S) = \bigcup_i NFP_i(S)$.

Tanım 3 *MAX-A: İfadenin her terimi için ilgili* UTP_i *ve* $NFP_{i,\bar{j}}$ *kümelerindeki tüm noktalar seçilir. Boole ifadesi için MAX-A kümesi : MAX-A(S)=* $UTP(S) \cup NFP(S)$.

Tanım 4 *İlişkili özgün doğruluk noktası ve yakın yanlışlık noktası ikilisi (CUTPNFP): $UTP_i(S)$ ve $NFP_{i,\bar{j}}(S)$ kümelerinden sırasıyla $\overrightarrow{u}$ ve $\overrightarrow{v}$ test girişi ikilisi seçilir. Bu ikilinin arasındaki tek fark, j indisi ile belirtilen değişkenin aldığı doğruluk değeri olmalıdır. Bu işlem olası tüm j değerleri ve tüm terimler için gerçekleştirilerek $CUTPNFP(S)$ kümesi elde edilir. Bu kümenin işlem referans hatası (İRH), değişken olumsuzlama hatası (DOH) ve ifade olumsuzlama hatası (İOH) sınıflarındaki hataları yakalaması amaçlanır.*

3.3 Myers'ın Yöntemi

Myers'ın kitabında tanıttığı neden-sonuç çizgelerinden test girişleri üretme yöntemi aşağıda verilmektedir [2]:

1. Bir sonuç düğümü seçilir ve doğru değer aldığı varsayılarak çizge üzerinde geriye doğru gidilir.
2. Seçilen düğümü doğru yapan tüm kombinasyonlar alınır. Kombinasyonlar alınırken:
   (a) Düğüm "VEYA" ilişkisi ile bağlıysa ve doğru değer alması gerekiyorsa, bu düğüme giden düğümlerden aynı anda en fazla birine doğru değer atanır. (Aynı anda tek bir değişkenin etkisini görebilmek için).
   (b) Düğüm "VE" ilişkisi ile bağlıysa ve yanlış değer alması gerekiyorsa,
   - her düğümün yanlış değer aldığı tek bir durum seçilir,
   - en az bir düğümün yanlış değer aldığı durumlarda, her düğüm için doğru değer aldığı tek bir durum seçilir

Myers'ın yönteminde diğer iki yöntemdeki gibi ANF Boole ifadeye ihtiyaç yoktur, çizge üzerinden ya da genel formdaki Boole ifadeden test girişleri üretilebilir.

4 Mutasyon Test Analizi

Oluşturulan test kümelerinin hata yakalama yeteneklerini karşılaştırmak için mutasyon analizi yöntemi oldukça sık kullanılmaktadır. Mutantlar asıl Boole ifade üzerinde, yazılım geliştiricilerin sık yaptığı hatalara dayalı bilgiden faydalanarak oluşturulan belli kurallar çerçevesinde yapılan küçük değişiklikler ile elde edilir. Yapılan değişikliklerin tipine göre, oluşturulan mutant, literatürde tanımlanan bir veya daha fazla hata sınıfını temsil eder. Bir test kümesi, bir mutant üzerinde çalıştırıldığında hatayı yakalayabiliyorsa o mutantı yok etmiş olarak kabul edilir. Test kümesinin başarısı yok edebildiği mutant sayısı ile ölçülür. İyi bir test üretim yönteminde amaç, olabilecek en az test girişi sayısı ile, en fazla mutantı yok etme başarısını elde etmektir.

Örnek bir $S = (c0 + (c1c2))c3$ Boole ifadesi üzerinde temel hata tipleri aşağıdaki gibi tanımlanabilir [25]:

İRH İşlem Referans Hatası. "VE" işlemi "VEYA" ile veya "VEYA" işlemi "VE" ile değiştirilir. Örn. $(c_0 + (c_1c_2)) + c_3$.

İOH İfade Olumsuzlama Hatası. İfadenin bir kısmı (alt ifade) olumsuzlanır. Örn. $(c_0 + (c_1c_2)')c_3$.

DOH Değişken Olumsuzlama Hatası. Boole ifadedeki değişkenlerden biri olumsuzlanır. Örn. $(c_0 + (c_1'c_2))c_3$.

EDH Eksik Değişken Hatası. Değişkenlerden birinin unutulması durumudur. Örn. $(c_0 + (c_1c_2))$.

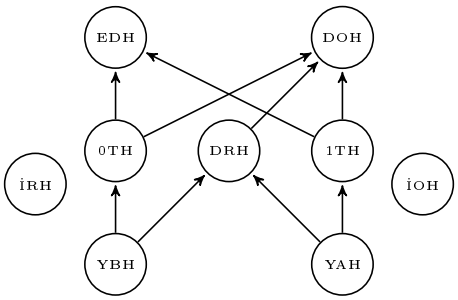
DRH Değişken Referans Hatası. Değişkenlerden biri ifadede yer alan başka bir değişken ile yer değiştirilir. Örn. $(c_1 + (c_1c_2))c_3$.

YBH Yantümce Birleşme Hatası. Değişken a yerine ab yazılması durumudur. Örn. $(c_0 + (c_1c_3c_2))c_3$.

YAH Yantümce Ayrılma Hatası. Değişken a yerine $a + b$ yazılması durumudur. Örn. $(c_0 + c_3 + (c_1c_2))c_3$.

0-T-H 0'a Takılma Hatası. Değişkenlerden birinin hep 0 olması durumudur. Örn. $(0 + (c_1c_2))c_3$.

1-T-H 1'e Takılma Hatası. Değişkenlerden birinin hep 1 olması durumudur. Örn. $(c_0 + (1 \cdot c_2))c_3$.

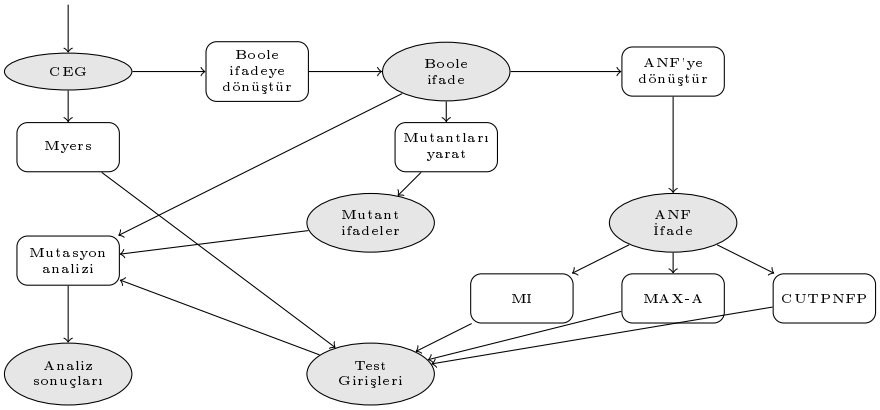


Şekil 3. Hata sınıf hiyerarşisi

Bütün hata sınıflarına uygun mutantlar oluşturulduktan sonra, bu mutantların her biri üzerinde test takımını tekrar tekrar çalıştırmak büyük programlarda, zaman ve kaynakların harcanması açısından anlamlı olmayabilir. Kuhn'un [26] çalışmasına göre, bir hata sınıfı başka bir hata sınıfının yaratabileceği tüm hatalı ifadeleri kapsayabiliyorsa o sınıf, diğerinden güçlü olarak kabul edilir. Bu çalışmada İOH'un DOH'a göre, DOH'un ise DRH'ye göre daha güçlü olduğu bir hiyerarşik yapı teorik olarak gösterilmiştir.

Tsuchiya ve Kikuno [27], Kuhn'un [26] çalışmasındaki hiyerarşiye ek olarak eksik ifade hatası sınıfını da eklemiştir. Lau ve Yu [28], bu hiyerarşiyi terim ve değişken hatalarının ilişkilerini analiz ederek daha da genişletmiştir.

Tüm bu çalışmalarda analiz edilen Boole ifadelerinin ANF olması gerekmektedir. Ancak, Kapoor ve Bowen [29], hiyerarşik analizi ve yeni hata sınıfları da ekleyerek, genel Boole ifadeleri için de geçerli olacak şekilde düzenlemiştir. Son olarak da, Chen ve arkadaşları [30], Kapoor ve Bowen'in [29] çalışmasını genişleterek, Şekil 3'te verilen hata sınıf hiyerarşisini sunmuştur. Bu yapıya göre, dokuz adet hata sınıfından yalnızca dört tanesini ele almak yeterlidir: İRH, YBH, YAH ve İOH.



Şekil 4. Neden-sonuç çizgelerinden test girişi üretme yöntemi

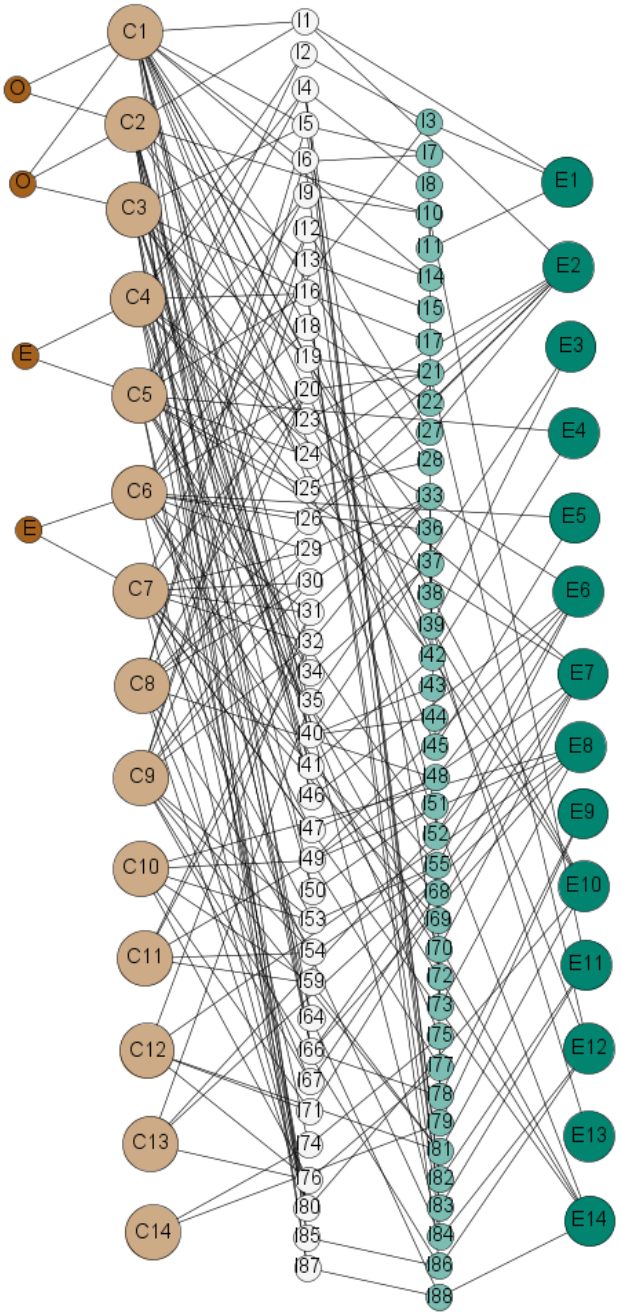
5 Neden-Sonuç Çizge Modeli ve Geliştirilen Araç Mimarisi

Neden-sonuç çizgesinin modellenmesi için GraphML [31] isimli, çizge tanımlamak için hazırlanmış, xml tabanlı bir dosya formatı kullanılmıştır. GraphML yapısı içerisindeki düğüm ve kenar özelliklerine ek olarak başka özellikler tanımlanabilmektedir. Neden-sonuç çizgelerini tanımlayabilmek için düğümlere; seviye (çizim sırasında neden-ara düğüm-sonuç ilişkisini korumak için), ilişki (düğümü oluşturan Boole ifadesini tanımlamak için), düğüm tipi (neden, sonuç, ara düğüm, koşul düğümü), koşul sonuç düğümü bilgisi (koşul düğümlerinin bağlı oldukları ara düğüm veya sonuç düğümü bilgisini tutmak için) tanımlanmıştır. Kenarlara ise yalnızca olumsuzluk bilgisi eklenmesi yeterli olmuştur.

Neden-sonuç çizgelerinin çizilmesi ve .graphml formatında bir çıktısının alınabilmesi ya da oluşturulan bir .graphml formatındaki çizgeyi çizdirmek için Gephi [32] isimli, açık kaynak kodlu kütüphane ve ona ait masaüstü programı kullanılmıştır. Java tabanlı Gephi kütüphanesi aracılığı ile .graphml dosyasından alınan çizge bilgileri, yine Java ile geliştirilen araçta oluşturulan model ile neden-sonuç çizge modeline dönüştürülmüştür.

Tüm sonuç düğümleri için ayrıca çizge ilişki bilgilerinden yola çıkarak Boole ifadeleri oluşturulmuştur. Boole ifadeleri MI, MAX-A ve CUTPNFP yöntemlerinde kullanılmak üzere kanonik ANF'ye dönüştürülmüştür. Bu dönüşüm için doğruluk tablosu (truth table) aracılığı ile dönüşüm algoritması kullanılmıştır. Bu algoritmada; her sonuç düğümü için, değişkenler için olası tüm değer kombinasyonlarına bakılır, sonucu doğru yapan satırlar seçilir. Bu satırlar seçilirken birbiri ile herhangi bir koşul (dışlayan veya, içeren veya, gerektirme vs.) ile bağlı olan değişkenlerin aldığı değerler ayrıca kontrol edilir, koşula aykırı olan bir durum varsa o kombinasyon seçilmez. Bu kombinasyonlarda doğru değerini alan değişkenler ve yanlış değeri alan değişkenlerin değilleri 've' bağlacı ile bağlanır. Bu kombinasyonlardan oluşturulan terimler daha sonra 'veya' bağlacı ile bağlanarak orijinal Boole ifadeye ilişkin olan kanonik ANF Boole ifadesi oluşturulmuş olur. Mutantları oluşturmak için, farklı hata tiplerine göre ayrı ayrı mutant oluşturma modelleri kurulmuştur. Sonuç düğümleri için oluşturulmuş olan orijinal

Boole ifadeleri üzerinden, istenen hata tipine göre oluşturulan bu modeller kullanılarak mutantlar oluşturulmuştur. Bu yöntem Şekil 4'te gösterilmektedir.



Şekil 5. TCAS-II'ye ait 14 gereksinimin çizilen neden-sonuç çizgesi

6 Değerlendirme ve Sonuçlar

TCAS-II, birçok hava aracı tipiyle uyumlu olan bir çarpışma önleme sistemidir ve literatürde test ile ilgili çalışmalarda çok sık kullanılmıştır [28, 33, 24, 6, 16]. TCAS-II'ye ait 14 gereksinim Şekil 5'teki neden-sonuç çizgesi olarak modellenmiş ve bu çizgeyi ifade eden graphml dosyası hazırlanmıştır. Neden-sonuç çizgesi Boole ifadelere çevrilerek, bu ifadeler Bölüm 5'te tanımlanan yöntem ile kanonik ANF'ye çevrilmiştir. Kanonik ANF'ye dönüştürülen ifadeler üzerinden MI, MAX-A ve CUTPNFP yöntemleri ile test kümeleri oluşturulmuştur.

Bölüm 4'te açıklanan mutasyon analizini uygulamak için, DOH, İOH, İRH, 0-TH, 1-TH, DRH, YBH, YAH hata sınıflarından olası tüm tekli (aynı anda tek bir değişiklik yapılarak) mutantlar oluşturulmuştur. Mutantlar üzerinde test takımları çalıştırılarak yakalanan mutant sayıları ayrı ayrı hata sınıfları ve üç ayrı yöntem için listelenmiştir. Tüm Boole ifadeler üzerinde sonuçları elde etmek için, açık kaynak kodlu bir Boole ifade değerlendirme kütüphanesi olan JBooleanExpression [34] kullanılmıştır.

Yapılan deney sonucu elde edilen sonuçlardan sonra: Tablo 1'de yöntemler sonucu oluşan toplam test giriş sayıları ve Tablo 2'de oluşturulan toplam mutant sayıları verilmiştir. Tablo 3'de farklı yöntemlerle oluşturulmuş olan test girişlerinin oluşan mutantlar üzerindeki hata yakalama başarısı verilmiştir. Son olarak da, Tablo 4'de bu başarıların toplam mutantlar üzerinde oranlanması ile elde edilen yüzdeler verilmiştir.

	MI	MAX-A	CUTPNFP	Myers
Test Sayısı	2125	4370	59	708

Tablo 1. Oluşan test giriş sayıları

	DOH+İOH	İRH	0-TH+1-TH	DRH	YBH	YAH	Tüm Sınıflar
Mutant Sayısı	391	238	504	238	252	252	1889

Tablo 2. Oluşan mutant sayıları

	Yakalanan Mutant Sayısı						
	DOH+İOH	İRH	0-TH+1-TH	DRH	YBH	YAH	Tüm Sınıflar
MI	270	176	254	146	99	91	1034
MAX-A	344	214	438	215	220	218	1649
CUTPNFP	267	168	150	130	84	64	863
Myers	267	173	252	148	92	86	1018

Tablo 3. Mutant yok etme sayıları

	Yakalanan Mutant Yüzdesi						
	DOH+İOH	İRH	0-TH+1-TH	DRH	YBH	YAH	Tüm Sınıflar
MI	0.69	0.74	0.50	0.61	0.39	0.36	0.55
MAX-A	0.88	0.90	0.87	0.90	0.87	0.87	0.87
CUTPNFP	0.68	0.71	0.30	0.55	0.34	0.25	0.46
Myers	0.68	0.73	0.50	0.62	0.36	0.34	0.54

Tablo 4. Mutant yok etme yüzdeleri

Oluşan toplam test giriş sayıları değerlendirildiğinde; MI ve MAX-A, kanonik ANF'deki tüm terimler ve tüm terimlerin değişkenlerini tek tek değerlendirdiği için, Myers'a göre daha fazla test oluşturmaktadır.

Sonuçlar başarı yüzdeleri üzerinden değerlendirildiğinde, tüm hata tipleri için: MAX-A, MI, Myers ve CUTPNFP olarak sıralanmıştır. Myers ile MI arasındaki başarı farkının çok düşük olmasının sebebi, MI yönteminde UTP seçimlerinin rastgele yapılmış olmasıdır. MI ve Myers'ın başarı yüzdeleri arasındaki fark, MI'daki rastgele seçimden dolayı az olabilse de, MI'ın kapsadığı test girişi daha fazla olduğu için MI Myers yerine tercih edilebilir. Kısıtlı bir zaman ve kaynak var ise, test giriş sayıları arasındaki farktan dolayı Myers, MI yerine tercih edilebilir. Bu noktada MI yönteminin uygulanabilmesi için Boole ifadeye dönüştürme, oluşan Boole ifadeyi kanonik ANF'ye dönüştürme gibi aşamaların masrafları ayrıca düşünülmelidir. CUTPNFP, yakalamayı amaçladığı DOH,İOH ve İRH sınıflarında çok daha az sayıda test girişi ile MI ve Myers'ın değerlerini yakalamıştır.

Bu çalışmaya ek olarak, Bölüm 1'de bahsedilen farklı test girişi üretme yöntemlerinden BOR ve MC/DC'nin de uygulamalarının yapılması, araca eklenmesi ve aynı analizden geçirilerek karşılaştırma tablosuna eklenmesi planlanmaktadır.

Kaynakça

1. Elmendorf, W. R.: Cause-Effect Graphs on Functional Testing. TR-00.2487, IBM Systems Development Division, Poughkeepsie, NY (1973).
2. Myers, G.: The Art of Software Testing, Second edition, (2004).
3. Paradkar, A., Tai, K. C., Vouk, M. A.: Automatic Test-Generation for Predicates. IEEE Transactions on Reliability, vol.45, no.4, 515–530 (1996).
4. Srivastava, P.R., Patel, P., Hatrola, S.: Cause Effect Graph to Decision Table generation. ACM SIGSOFT Software Eng.Notes, vol.34, no.2 (2009).
5. Matur, A. P.: Software testing. 1st edition, Pearson Publication (2008).
6. Weyuker, E., Goradia, T., Singh, A.: Automatically Generating Test Data from a Boolean Specification. IEEE Transactions on Software Engineering, vol.20, no.5, 353-363 (1994).
7. Chen, T., Lau, M., Yu, Y.: MUMCUT: A fault-based strategy for testing boolean specications. In: Asia-Pacic Software Engineering Conference, 606, IEEE Computer Society, Los Alamitos, CA, USA, (1999).
8. Chen, T.Y., Lau, M.F.: Test case selection strategies based on Boolean specifications. Softw. Test. Verif. Reliab. vol.11, 165–180 (2001).
9. Paradkar, A., Tai, K. C., Vouk, M. A.: Specification-Based Testing Using Cause-Effect Graphs. Annals of Software Engineering, vol.4, 133–157 (1997).
10. Vilkomir, S. A., Bowen, J. P.: Reinforced Condition/Decision Coverage (RC/DC): A New Criterion for Software Testing. In 2nd International Conference, Formal Specification and Development in Z and B, volume 2272 of Lecture Notes in Computer Science, 295–313. Springer-Verlag, (2002).
11. Chen, T.Y., Grant, D.D., Lau, M.F., Ng, S.P., Vasa, V.R.: BEAT:Boolean Expression Fault-Based Test Case Generator. In: Information Technology:Research and Education Conference, 625-629, (2003)
12. Kaminski,G. K. and Ammann, P.: Using Logic Criterion Feasibility to Reduce Test Set Size While Guaranteeing Fault Detection. In: ICST'09: Proceedings of the 2nd International Conference on Software Testing Verification and Validation. Washington, DC, USA: IEEE Computer Society, Apr. 1–4, 356–365, (2009).
13. Singh, R.K., Chandra, P., Singh, Y.: An Evaluation of Boolean Expression Testing Techniques. ACM SIGSOFT Software Engineering Notes, vol.31, no.5, 1-6 (2006).

14. Ayav, T., Belli, F.: Boolean Differentiation for Formalizin Myers' Cause-Effect Graph Testing Technique. In: Software Quality, Reliability and Security-Companion, IEEE, 138-143, Vancouver, BC (2015).
15. Fraser, G., Gargantini,A.: Generating minimal fault detecting test suites for Boolean expressions. In: Software Testing, Verification, and Validation Workshops, 37-45, Paris (2010).
16. Gargantini,A., Fraser,G.: Generating minimal fault detecting test suites for general Boolean specifications. Information and Software Technology, vol.53, no.11, 1263–1273 (2011).
17. Sziray, J.: Evaluation of boolean graphs in software testing. In Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on, 225-230, IEEE, (2013).
18. Vilkomir, S., Starov, O., Bhambroo, R.: Evaluation of t-wise approach for testing logical expressions in software. In Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on. 249-256, IEEE, (2013).
19. Paul, T. K., Lau, M. F.: A systematic literature review on modified condition and decision coverage. Proceedings of the 29th Annual ACM Symposium on Applied Computing, 1301-1308, ACM, (2014).
20. Chung, I.: Investigating effectiveness of software testing with cause-effect graphs. International Journal of Software Engineering and Its Applications, vol.8, no.7, 41-54 (2014).
21. Chung, I.: CEGPairGen: an automated tool for generating pairwise tests from cause–effect graphs. International Journal of Software Engineering and Its Applications, vol.9, no.1, 53-66 (2015).
22. Chung, I.: Modeling Pairwise Test Generation from Cause-Effect Graphs as a Boolean Satisfiability Problem. International Journal of Contents. vol.10, no.3, 41-46 (2014).
23. Sun, C., Zai, Y., Liu, H.: Evaluating and Comparing Fault-Based Testing Strategies for General Boolean Specifications: A Series of Experiments. The Computer Journal, vol.58, no.5, 1199-1213 (2015).
24. Badhera, U., G.N. P., Taruna, S.: Fault Based Techniques for Testing Boolean Expressions : A Survey. International Journal of Computer Science & Engineering Survey. vol.3, no.1, 81–90 (2011).
25. Ayav, T.: Yazılım Yapısal Kapsama Analizinde Testlerin Önceliklendirilmesi. 9th UYMS , İzmir (2015).
26. Kuhn, R.: Fault classes and error detection capability of specification-based testing. ACM Trans. Softw. Eng. and Methodol., vol.8, no.4, 411–424 (1999).
27. Tsuchiya, T., Kikuno, T.: On fault classes and error detection capability of specification-based testing. ACM Trans. Softw. Eng. and Methodol., vol.11, no.1, 58–62 (2002).
28. Lau, M.F., Yu, Y.T.: An extended fault class hierarchy for specification-based testing. ACM Trans. Softw. Eng. and Methodol., vol.14, no.3, 247-276 (2005).
29. Kapoor, K., Bowen, J.P.: Test conditions for fault classes in Boolean specifications. ACM Trans. Softw. Eng. Methodol., vol.16, no.3, 10 (2007).
30. Chen, Z., Chen, T.Y., Xu, B.: A Revisit of Fault Class Hierarchies in General Boolean Specifications. ACM Trans. Softw. Eng. Methodol., vol.20, no.3, 13:1-13:11 (2011).
31. The GraphML File Format, `http://graphml.graphdrawing.org/`
32. Gephi: The Open Graph Viz Platform, `https://gephi.org/`
33. Chen,T.Y., Lau,M.F., Sim,K.Y., Sun,C.a.: On detecting faults for boolean expressions. Software Quality Journal, vol.17, no.3, 245–261, (2009).
34. JBooleanExpression: Java Boolean Expression Evaluator, `http://jboolexpr.sourceforge.net/`