Birleşik Kombinezon Etkileşim Sınama Yöntemi

Hanefi Mercan ve Cemal Yılmaz
Mühendislik ve Doğa Bilimleri Fakültesi, Sabancı Üniversitesi, İstanbul, Türkiye
{hanefimercan,cyilmaz}@sabanciuniv.edu

Özet. Günümüz yazılım sistemlerinin test edilmesi neredeyse her zaman değişkenlik uzaylarının (örn. girdi uzayının ve/veya konfigürasyon uzayının) örneklenmesi ve sadece seçilen konfigürasyonların test edilmesi şeklinde gerçekleştirilir. Kombinezon Etkileşim Sınama (KES) yöntemleri bu örneklendirmelerin sistematik bir şekilde gerçekleştirilmesini sağlar. Bu yöntemler yardımıyla öncelikle verilen bir konfigürasyon uzayı ve kapsama kriteri altında "maaliyet-etkili" bir yolla tam bir kapsama elde edilecek (diğer bir deyişle kapsama kriterinin belirttiği bütün kombinasyonlar kapsanacak) şekilde bir KES objesi üretilir. Bu KES objesi konfigürasyon uzayından seçilmiş bir konfigürasyon kümesini ifade eder. Daha sonra sistemin sınanması ise KES objesindeki konfigürasyonların test edilmesi ile gerçekleştirilir. Bizim bu çalışmadaki esas amacımız; KES yöntemlerinin pratik hayatta kullanılabilirliğini daha da arttırmak adına yazılımcıların kendi ihtiyaçları doğrultusunda özelleştirdikleri özgün konfigürasyon modellerini, özgün kapsama kriterlerini ve özgün KES objeleri tanımlamalarını mümkün kılmaktır. Bu çalışmamızda sunmuş olduğumuz hesaplama tekniği verilen bir konfigürasyon uzayı ve kapsama kriteri tanımlarını girdi olarak alıp, istenilen KES objesini etkili ve verimli bir şekilde hesaplayacak hesaplama yöntemlerini içermektedir. Biz bu yöntemi Birleşik Kombinezon Etkileşim Sınama Yöntemi (Unified Combinatorial Interaction Testing), kısaca B-KES, olarak adlandırıyoruz. Bu yöntem sayesinde hali hazırda var olan geleneksel KES objeleri, B-KES objelerinin özel bir durumu olarak ifade edilebileceği gibi bundan önce görülmemiş türlerdeki KES objeleri de tanımlanabilecek ve de tanımlanan özgün objeler, tanımlarından bağımsız bir şekilde birleştirilmiş (unified) bir yöntemle hesaplanabilecektir.

Anahtar Kelimeler: Kombinezon etkileşim sınama, kapsayan diziler, kısıt tatmin problemleri

Unified Combinatorial Interaction Testing

Abstract. Testing software systems almost always involve sampling enormous variability spaces, such as input and configuration spaces, and testing representative instances of a system's behavior. This sampling is commonly performed with techniques collectively referred to as combinatorial interaction testing (CIT). These techniques typically have two inputs: configurations spaces and coverage crieteria. Configuration space contains all valid option-value

combinations (configurations). On the other hand coverage_criteria defines all valid combinations which needs to be tested. Under given configuration space and coverage criteria, an CIT object is computed as an output. This CIT object which contains a sample of a subset of configuration space, is computed as "cost-effectively" and have full coverage under the given coverage criteria. After CIT objects are computed, quality assureance for software systems is done only testing chosen configurations. Our main goal in this work is further exploit the benefits of CIT to improve the efficiency, effectiveness and applicibilty of CIT approaches in practial scenarios by enabling practitioners to define their own space for testing as well as their own coverage criterion (thus enabling them to invent their own application-specific CIT objects). The proposed techinque in this work contains methods to compute a CIT object under given configuration space and coverage criteria. This approach will be referred as Unified Combinatorial Interaction Testing (U-CIT) in the rest of the paper. To this end, this requires truly flexible CIT approaches. In this work, we would like to develop tools for practitioners to define their own application-specific variability space as well as their own application.

Keywords: Combinatorial interaction testing, covering arrays, constraint satisfaction problems

1 Giriş

Günümüz yazılım sistemleri; kullanıcı girdileri, konfigürasyon parametreleri ve iş parçacıkları etkileşimleri (thread interleavings) gibi test edilmesi gereken bir çok değişkenlik parametresine sahiptir. Bu değişkenlik parametreleri son kullanıcılar için esneklik sağlarken yazılım geliştirenler için yazılımların kalite güvencesinin sağlanması noktasında ciddi sıkıntılara sebebiyet verir. Küçük çaptaki yazılımlar için dahi tüm değişkenlik uzayının test edilmesi genellikle mümkün değildir. Örneğin; Apache sunucusunun bir sürümü son kullanıcılar tarafından yapılandırılabilen 172 konfigürasyon parametresine ve bu parametrelerle oluşturulabilecek $1{,}8x10^{55}$ farklı konfigürasyona sahiptir [1]. Her bir konfigürasyonun test edilmesi bir saniye bile sürse bütün konfigürasyonların test edilmesi için gereken süre, büyük patlamadan (big bang) bu yana geçen süreden daha fazladır. Dolayısıyla Apache (ve benzeri diğer sistemler) için bütün konfigürasyonların test edilmesi söz konusu bile olamaz.

Bu ve benzeri sebeplerden dolayı endüstriyel sistemlerin test edilmesi nerdeyse her zaman çok büyük bir değişkenlik uzayından (ki bu tür uzaylar dokümanın geri kalan kısmında konfigürasyon uzayları olarak adlandırılacaktır) örnekleme yöntemiyle seçilen ve tüm uzayı temsil kabiliyetine sahip olduğu düşünülen bir alt uzay kullanılarak gerçekleştirilir. Pratik hayatta bu örnekleme işlevi genellikle *Kombinezon Etkileşim Sınama (KES)* yöntemleri adı altında toplanmış teknikler kullanılarak gerçekleştirilir [2]. KES yöntemlerinin başlıca iki girdisi vardır: *konfigürasyon uzayı* ve *kapsama kriteri*. Konfigürasyon uzayı, geçerli bütün parametre değerleri kombinasyonlarını (konfigürasyonları) içerir. Kapsama kriteri ise test edilmesi istenilen bütün geçerli kombinasyonları tanımlar. Verilen bir konfigürasyon uzayı ve kapsama kriteri için çıktı olarak bir *KES objesi* hesaplanır. Bu obje; kapsama kriteri altında "maaliyet-

etkili" bir yolla tam bir kapsama elde edilecek şekilde konfigürasyon uzayından seçilmiş bir konfigürasyon kümesinden oluşur. KES objesi hesaplandıktan sonra yazılım sisteminin sınanması sadece seçilen konfigürasyonların test edilmesi ile sağlanır. KES objelerine örnek olarak sıklıkla kullanılan *t li kapsayan diziler* (t-way covering array) verilebilir. *t* li kapsayan diziler, konfigürasyon uzayını kapalı bir şekilde (implicit) tanımlayan bir konfigürasyon uzayı modelini girdi olarak alır. En basit haliyle bu model, her biri sonlu değer alan konfigürasyon parametrelerini ve bu parametrelerin alabileceği değerleri içerir. Verilen bir konfigürasyon uzayı modeli için *t* li kapsayan bir dizi, konfigürasyon parametreleri kümesinin her *t* li altkümesi için, ilgili parametre değerlerinin her bir kombinasyonunu en az bir kere içerecek şekilde oluşturulmuş bir konfigürasyon kümesidir. KES yöntemlerini kullanmanın temel gerekçesi; bu yöntemlerin *t* veya *t*'den daha az sayıda parametrenin etkileşiminden kaynaklanan hataların (belirli kabuller dahilinde) etkili ve verimli bir şekilde bulunmasına olanak sağlamasıdır. Literatürde yer alan birçok çalışma KES yöntemlerinin farklı uygulama alanlarında başarılı bir şekilde kullanıldığını göstermektedir [5,6,7,8,9,10,11].

İlk zamanlarda KES yaklaşımları sadece iki değer alan parametrelerin ikili ilişkilerini test etmek amacıyla kullanılıyordu. Daha sonra pratisyenler (örn. yazılım mühendisleri) üç değer alan parametreler için kapsayan dizilere ve sonrasında ise karışık kapsayan dizilere; yani her parametrenin farklı sayıda değer alabileceği kapsayan dizilere, ihtiyaç duymuşlardır. Bu ihtiyaçları karşılayabilmek adına KES alanında yeni çalışmalar yapılmaya başlanmıştır. Bunları takiben yüksek kapsama kuvvetine sahip kapsayan dizilerin icadı ise bu alanda yapılan bir başka yeniliktir. Ayrıca karışık kapsama kuvvetine (variable strength) sahip kapsayan dizilerde, yani farklı parametrelerin kendi aralarında farklı kapsama kuvvetlerine sahip olmaları, gün geçtikçe popüler olmaya başlamıştır. Bu alanda getirilen her yenilik "geleneksel olmayan" bu objeleri "geleneksel" hale getirebilmek için bunları üretebilecek ve hesaplayabilecek yeni algoritmaların araştırılmasına ve geliştirilmesine sebep olmuşlardır. Başka bir deyişle, KES alanında yapılan bütün çalışmalarda izlenen yöntem şu şekilde özetlenebilir: araştırmacılar sahadaki test senaryolarını desteklemek için özgün konfigürasyon uzayı modelleri ve/veya özgün kapsama kriterleri tanımlar, bu kriterleri sağlayacak özgün KES objeleri icat eder ve bu objeleri hesaplayacak özelleştirilmiş yöntemler geliştirir. Sahadaki pratisyenler ise geliştirilen KES objelerini kullanmak suretiyle yazılım sistemlerini test eder. Biz KES'in faydalarını "geleneksel olmayan" konfigürasyon uzaylarını "geleneksel olmayan" kapsama kriterleri kullanarak daha da ileriye götürebileceğimize inanıyoruz. Fakat bu düşünce ancak gerçekten esnek KES yaklaşımları ile mümkün olabilir. Bu nedenle, eğer pratisyenlerin kendi uygulamalarına özel konfigürasyon uzaylarını ve kapsama kriterlerini tanımlamaya olanak sağlayan araçlar olsaydı, KES'in esnekliği arttırılarak pratik hayatta uygulanabilirliğinin daha da arttırabileceğine inanıyoruz.

Bizim bu çalışmamızda sunmuş olduğumuz yaklaşımın amacı ise bu anlatmış olduğumuz akışı tersine çevirmek suretiyle KES yaklaşımlarının "geleneksel olmayan" konfigürasyon uzaylarında "geleneksel olmayan" kapsama kriterleri ile kullanılmasına olanak sağlamak suretiyle KES yaklaşımlarının verimliliklerini, etkinliklerini ve sahadaki uygulanabilirliklerini önemli ölçüde artırmaktır. Diğer bir deyişle; bu çalışmamızda önermiş olduğumuz yaklaşım sayesinde araştırmacıların yeni KES objeleri icat etmek suretiyle pratisyenlere neyin test edilmesi gerektiğini dikte ettirmeleri yerine pra-

tisyenlerin, kendi ihtiyaçları doğrultusunda özelleştirdikleri özgün KES objeleri tanımlamaları (diğer bir deyişle icat etmeleri) sağlanacaktır ki dokümanın geri kalan kısmında bu yaklaşım Birleştirilmiş Kombinezon Etkileşim Sınama Yöntemi (B-KES) olarak adlandırılmıştır (Unified Combinatorial Interaction Testing) [12]. Bu amaca ulaşmak gerçek anlamda esnek KES yaklaşımlarının geliştirilmesini gerektirmektedir. Önceki çalışmamızda [12] bu amaca ulaşmak adına ilginç bir kısıt çözüm (constraint solving) problemi üzerinde yoğunlaştık. Bu çalışmamızda ise B-KES'in test etme yönüne daha çok ağırlık verip, neden daha esnek bir yapıda olduğunu gösteren örnekler veriyoruz.

Makalenin 2. bölümünde B-KES'i genelleştirmek adına daha resmi tanımlarla anlatıyoruz. 3. bölüm KES alanında yapılan çalışmalar ile ilgili literatür özeti veriyor. 4. bölümde ise B-KES'in önemini anlatmak üzere örnekler verip aynı zamanda pratik hayatta B-KES'e duyulan ihtiyacı anlatıyoruz. 5. bölümde B-KES objeleri hesaplayabilmek için açgözlü (greedy) bir algoritma anlatıyoruz. Son olarak 6. bölümde ise sonuç kısmını veriyoruz.

2 Birleşik Kombinezon Etkileşim Sınama

Bu bölümde B-KES'i daha resmi bir şekilde anlatıp genelleştirmek adına aşağıdaki tanımları kullanıyoruz. Her yazılım sistemi için uygulanabilirliği farklı olduğundan tanımlar soyut bir şekilde verilmiştir.

- Bir *B-KES isterleri* (requirements): Bir B-KES isteri kısıtlarla ifade edilmiş birimlerdir.
- Bir *B-KES test durumu:* Bir B-KES test durumu birlikte test edilebilen isterler kümesidir. Başka bir deyişle; birlikte tatmin edilebilir (satisfiable) kısıtlar kümesidir. Fakat, tüm isterler kombinasyonları pratik hayatta her zaman geçerli olmayabilir.
- Bir *B-KES uzay modeli*: Bir B-KES uzay modeli tüm geçerli B-KES isterlerini kapalı bir şekilde tanımlayan kısıtlardan ve geçerli B-KES test durumlarından oluşan sistemdir.
- Bir *B-KES kapsama kriteri*: Bir B-KES kapsama kriteri kapsanması istenen B-KES isterlerini kapalı bir şekilde tanımlayan kriterdir.

Genel bağlamda, B-KES girdi olarak bir sistem modellemesi alır. Bu modelleme tüm geçerli test durumları uzayını ve test edilmek üzere kapsanması gereken tüm olası isterleri ifade eden kapsama kriterinden oluşmaktadır. Çıktı olarak ise verilen test kriteri altında tam kapsama elde eden test durumlarından oluşan bir test havuzu verir.

B-KES'i birleşik bir yaklaşım yapan şey kapsanması gereken birimlerin, test durumlarının ve test durumlarının üretileceği uzayın bir kısıt olarak ifade edilmesidir. Sonuç olarak, B-KES üretme problemi aslında büyük ve ilginç bir kısıt çözme problemine dönmektedir [12]. Şunu belirtmek isteriz ki, "kısıt" terimini genel anlamda kullanıyoruz; yani mantıksal programlama dillerinden bağımsız her hangi bir sınırlama olarak tanımlanabilen bir kısıt kastedilmektedir.

Bir kapsama kriteri aslında kısıtlar ile ifade edilmiş kapsanmak istenen birimlerin (hepsi birlikte olmak zorunda değil, grup halinde olabilir) oluşturduğu kümeyi tanımlar.

Daha sonra, bu kısıtların minimum sayıda alt kümeleriyle oluşturulan; öyle ki alt kümedeki tüm kısıtlamaların kendi aralarında tatmin edilebilir ve bütün bu alt kümelerin birleştirilmesiyle tüm kısıt kümesi tekrar oluşturulabilen, bir B-KES objesi üretilir. Bu kısıtlardan oluşan her alt küme geçerli bir test durumuna karşılık gelir; yani birlikte test edilebilen birimleri kapsayan bir test durumu. Bu yüzden, tüm bu alt kümeler kullanılarak kapsama kriteri altında B-KES'i oluşturan test durumları (her bir alt küme bir test) üretilir. Yani başka bir deyişle kısıtlar kümesini minimum sayıda alt kümelere bölerek kendi aralarında tatmin edilebilir alt kümeler elde etmek, B-KES için test durumları üretmek problemiyle aynıdır.

3 Literatür Özeti

Nie ve Leung [13] geleneksel kapsayan dizileri hesaplama yöntemlerini başlıca dört kategoriye ayırır: rastgele aramaya dayalı yöntemler, matematiksel yöntemler, aç gözlü (greedy) yöntemler ve lokal aramaya dayalı yöntemler. Rastgele aramaya dayalı yaklaşımlar yer değiştirme stratejisi uygulamayan bir rastgele arama yöntemi (random search without replacement) kullanırlar [4]. Bu yöntemde her adımda bir konfigürasyon, geçerli konfigürasyon uzayından rastgele seçilir. Tekrarlamalar, seçilen konfigürasyonlar bir *t* li geleneksel kapsayan dizi teşkil edene kadar devam eder. Geleneksel kapsayan dizileri hesaplamak için matematiksel yöntemler de kullanılmıştır [1,3,15]. Bu yöntemlerde büyük konfigürasyon uzayı modelleri için kapsayan dizileri hesaplarken bu modellerin küçük parçaları için hesaplanmış kapsayan dizileri, özyinelemeli (recursive) bir şekilde kullanır. Aç gözlü yöntemler ise tekrarlamalı bir şekilde çalışır [6,10]. Her tekrarlamada, aday olarak değerlendirilen bir konfigürasyon kümesinden, daha önce kapsanmamış *t* li parametre değerleri kombinasyonlarından en fazlasını kapsayan konfigürasyon, kapsayan diziye eklenir. Tekrarlamalar, istenilen bütün *t* li kombinasyonlar, seçilen konfigürasyonlar tarafından kapsandığında sona erer. Lokal aramaya (heuristic search) dayalı yöntemler [16, 17] ise geleneksel kapsayan dizileri hesaplamak için yapay zekâya dayalı teknikler kullanır. Bu yöntemler, ellerinde her an bir aday konfigürasyon kümesi bulundurur ve bu küme, geleneksel *t* li bir kapsayan dizi teşkil edene kadar tekrarlamalı bir şekilde bu kümeye çeşitli dönüşümler uygular.

Literatürde yer alan ve yukarıda özetlenen bu çalışmaların amacı geleneksel kapsayan dizileri hesaplamaktır. Farklı türde özgün kapsayan diziler olan T-KAD, M-KAD ve TM-KAD'ları üzerine de ayrıca çalışmalar yapılmaktadır. Literatürde, yakın geçmişteki bir yayınımız [18] dışında test durumlarına özel kısıtları göz önüne alarak kapsayan dizi hesaplayan bir çalışmaya rastlanmamıştır. Reel test maliyetlerini göz önüne alan ve makalemizde önerilen çalışmaya en yakın olan çalışmalar ise KES yaklaşımlarında test durumlarının önceliklendirilmesi ile ilgili olan çalışmalardır [19,20,21,22,23]. Elbaum konfigürasyonların kaynak kod kapsama yetilerini notlandırarak geleneksel kapsayan diziler tarafından seçilen konfigürasyonları önceliklendirmiştir. Qu ise önceliklendirme için konfigürasyonların hataları keşfetme yetilerini kullanmıştır. Srikanth ve Kimoto, konfigürasyonlar arasında geçiş yapmanın sebebiyet verdiği ek maliyetleri, kapsayan diziler tarafından seçilmiş konfigürasyonları önceliklendirmek için kullanmıştır. Konfigürasyon geçiş maliyeti, test esnasında, sistemin mevcut konfigürasyonunu bir sonraki konfigürasyona çevirmek için gerekli olan maliyettir. Bu tür maliyetler, genellikle test altındaki sistemin konfigüre edilmesi için insan

müdahalesinin gerekli olduğu ve oluşturulan konfigürasyonların gelecek kullanımlar için bütün veya kısmi olarak saklanamadığı durumlarda geçerlidir. Örneğin; test altındaki sistemin bir konfigürasyon parametresinin sistem ile entegre edilmiş bir yazıcıda kullanılan kağıt tipi olduğunu düşünelim. Yazıcıdaki kağıt tipini değiştirmek insan müdahalesi gerektirir ki bu maliyetli bir iştir. Ayrıca bir kağıt tipi ile konfigüre edilmiş yazıcı (birden fazla yazıcı kullanılmadığı sürece), gelecek kullanımlar için saklanamaz. Bu basit senaryoda konfigürasyon geçiş maliyetlerini minimize etmeyi amaçlayan yöntemler, yazıcıdaki kağıt tipi değiştirme sayısını minimize edecek şekilde seçilen konfigürasyonları sıralar.

Bu bölümde verilmiş tüm çalışmaların esas amacı özelleştirilmiş bir KES alanında, etkili ve/veya verimli bir şekilde KES objesi üretmektedir. Öte yandan B-KES'in amacı ise bunların hepsini üretebilecek yetenekte olmak ve son kullanıcılar tarafından belirlenmiş kapsama kriteri kullanılarak daha kompleks yapıda test havuzları oluşturmaya olanak sağlamaktır. Buna ilk adım olarak daha önce sunmuş olduğumuz çalışmamızda bu amacımıza ulaşabilmek için bir kısıt probleminden bahsetmiştik [12]. Bu çalışmamızda ise buna ek olarak B-KES'in bu alandaki önemini daha iyi anlatabilmek adına gerçek hayattan esinlenerek oluşturulmuş örnekler sunuyoruz.

4 B-KES Yöntemlerine Duyulan İhtiyaç

Bir KES objesi olan kapsayan diziler bu alanda test havuzu olarak sıklıkla kullanılan objelerdir. Fakat buna rağmen bu objelerin esneklikleri kısıtlıdır. Diğer bir deyişle bu objeler sadece bir tür konfigürasyon uzayı ve sadece bir tür kapsama kriteri kabul eder. Örneğin; kapsayan diziler için konfigürasyon uzayı sonlu değerler alabilen konfigürasyon parametrelerinden oluşur ve her konfigürasyonda her bir parametre geçerli bir değer almak zorundadır. Sahadaki ihtiyaçlar bu modele uymuyorsa geleneksel kapsayan dizilerden yararlanılamayabilir ya da bu dizilerin gelişi güzel kullanılması sonucunda sistemlerdeki hatalar yeterince etkili ve verimli bir şekilde tespit edilemeyebilir.

Örneğin; test edilmesi gereken sistemin yüksek seviyede yapılandırılabilir bir sistem olduğunu varsayarsak, bu sistemi test etmek için *t* li bir geleneksel kapsayan dizi hesaplandıktan sonra seçilen konfigürasyonlarda sistem için yazılmış olan bir dizi test durumunun koşturulması gerektiğini, fakat her test durumunun her konfigürasyonda test durumuna özel bir takım kısıtlardan dolayı çalışamadığını varsayalım. Bu senaryoda geleneksel kapsayan dizilerin kullanılması *maskeleme etkileri* olarak adlandırılan ve test edilmemiş parametre değerleri kombinasyonu ve test durumları ikililerini test edilmiş gibi göstermek suretiyle yazılım sınama süreçlerine zarar veren etkilerin oluşmasına sebebiyet verecektir [14]. Bunun sebebi geleneksel kapsayan dizilerin test durumlarından ve test durumlarına özel kısıtları dikkate almamasıdır. Dolayısıyla bir test durumunun, kendisine özel bir kısıtı sağlamayan bir konfigürasyon üzerinde koşturulması istenebilir. Bu durumda test koşturulamayacağından, test durumu bahse konu konfigürasyon içerisinde yer alan ve kendisi için geçerli olan hiç bir parametre değerleri kombinasyonunu test edemeyecektir (bu kombinasyonlar maskelenmiş olacaktır). Bu durumu engellemek için geleneksel kapsayan dizilerin girdi olarak aldığı konfigürasyon uzayı modeli, test durumları ve test durumlarına özel kısıtları alacak şekilde genişletilmeli, her test durumu için geçerli olan *t* li parametre değerleri kombinasyonları dik-

kate alınarak kapsama kriteri güncellenmeli ve sonuçta oluşacak olan yeni KES objelerini hesaplamak için hesaplama yöntemleri geliştirilmeli ve gerçeklenmelidir. Yılmaz et. al [18] bütün bu değişiklikleri adreslemek suretiyle *test durumlarını dikkate alan kapsayan diziler* adı altında özgün KES objeleri tanımlamış ve bu çalışma yazılım mühendisliği alanında en prestijli dergilerden birinde, geleneksel kapsayan diziler yazılım testleri için kullanılmaya başlandıktan onlarca sene sonra ve sadece geleneksel kapsayan dizileri hesaplamak için 50'den fazla makale varken [13] yayımlanabilmiştir. Fakat sahada karşılaşabilecek her bir özel senaryo için bu yeni KES objesi geliştirme sürecinin baştan işletilmesi sürdürülebilir değildir. Dolayısıyla bu çalışmamızda önermiş olduğumuz B-KES yaklaşımına şiddetle ihtiyaç vardır.

Bu iddialarımızı daha net bir şekilde gösterip B-KES'in esnekliğini daha detaylı gösterebilmek için 2 tane örnek sunuyoruz.

Örnek 1: Şekil 1'de verilmiş olan sistem daha önce kullanmış olduğumuz yapılandırılabilirlikleri yüksek ve sıklıkla kullanılan bir veritabanı yönetimi sistemi olan MySQL ve bir HTTP sunucusu olan Apache yazılım sistemlerinden esinlenerek oluşturduğumuz örnektir. Bu örnekte, her biri 2 değer alabilen, *D* ve *Y* (Doğru ve Yanlış), sistem kurulmadan önce test edilmesi gereken 5 tane derleme zamanı konfigürasyon parametresi (p_1, p_2, p_3, p_4 ve p_5) bulunmaktadır. Şekil 1'de bir örneği verildiği üzere, C ve C++ gibi ön işleme (preprocessing) gereken programlama dillerinde bu parametreler genellikle #ifdef gibi ön işlem direktifleri kullanılarak yazılmaktadır.

Şekil 1 iç içe geçmiş if durumlarını tanımlayan derleme zamanı konfigürasyon parametrelerinin birbirleriyle nasıl ilişkide olduğunu gösteren bir sistemi örneklendirmektedir. Bu tip sistemlerde pratisyenlerin sıklıkla kullanmayı tercih ettikleri yapısal kapsama sunan test yöntemlerinden birisi *karar kavrama (KK)* (decision coverage) kriteridir. KK kriterinde tam bir kapsama elde edebilmek için mümkün olan tüm kararlar için bütün sonuçların denenmesi gerekir. Örneğin, Şekil 1'de verilmiş olan iç içe geçmiş #ifdef bloklarında %100 KK kapsaması elde etmek için dıştaki kararın (p_1 && p_2 && p_3) ve dıştaki bu karar ile korunmuş olan içteki kararın (p_4 || p_5) doğru ve yanlış olarak en azından bir kere test edilmeleri gerekir.

```
#ifdef (p1) ... #endif
#ifdef (p2) ... #endif
#ifdef (p3) ... #endif
#ifdef (p4) ... #endif
#ifdef (p5) ... #endif

#ifdef (p1 && p2 && p3)
  ...
     #ifdef (p4 || p5)
       ...
     #endif
  ...
#endif
```

Şekil. 1. Önişlemci direktifleri kullanılarak 5 tane ön derleme parametresinden oluşan varsayımsal bir sistem.

Ayrıca bir pratisyenin tam bir KK kapsaması elde etmek dışında tüm bu parametrelerin ikili ilişkilerini de test etmek istediğini varsayalım. Bunu yapmak için 2 li bir kapsayan dizisi (Şekilde 2'deki ilk beş sütun) hesaplanmış ve sonrasında bu kapsayan dizinin konfigürasyonları kullanılarak test edilmiştir. Başka bir deyişle, Şekil 2'deki ilk beş sütun Şekil 1'de verilmiş olan parametrelerin alabileceği değerleri göstermektedir. Son iki sütun ise #ifdef bloklarındaki iç içe geçmiş kararların sonuçlarını göstermektedir: doğru için *'D'*, yanlış için *'Y'* ve eğer bir karar dıştaki karar tarafından korunduğundan dolayı hiç işlenemediyse '-' kullanılmıştır. Fakat, Şekil 2'de verilmiş olan 2 li kapsayan dizisi Şekil 1'de verilmiş olan ilk beş #ifdef blokları için %100 KK kapsaması elde ederken, iç içe geçmiş #ifdef durumları için 4 ihtimalden sadece 3'ünü kapsayarak %75 KK kapsaması elde etmektedir. Kapsanmamış durum ise içteki kararın $(p_3 \parallel p_4)$ yanlış olarak hesaplanmasıdır. Bu durumun kapsanması sadece 5'li $(p_1$=D, p_2=D, p_3=D, p_4=Y, p_5=Y) kombinasyonu kullanılarak mümkün kılınabilir.

p_1	p_2	p_3	p_4	p_5	$p_1 \wedge p_2 \wedge p_3$	$p_4 \wedge p_5$
D	Y	Y	D	Y	*Y*	-
D	D	D	Y	D	*D*	*D*
Y	Y	D	D	D	*Y*	-
Y	D	Y	Y	Y	*Y*	-
Y	D	D	D	Y	*Y*	-
Y	Y	Y	Y	D	*Y*	-

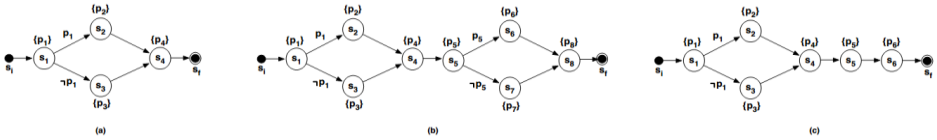
Şekil. 2. Şekil 1'deki senaryo için oluşturulmuş 2 li kapsayan diziye bir örnek ve kararlardan elde edilmiş sonuçlar.

Başka bir alternatif çözüm yolu ise kapsayan dizinin kapsama kuvvetini arttırmak olabilir. Ama bu yaklaşımda test havuzunun boyutunu çok fazla arttırabilir. Örneğin, KK kriteri için tam kapsama elde etmeyi garanti etmek için eklenmesi gereken kombinasyon beş tane parametre içerdiğinden, 5 li bir kapsayan dizi kullanmak gerekecektir. Yani 5 uzunluğundaki bu özelliklerde bir kapsayan dizi 32 tane konfigürasyon içerecek olup aslında tüm konfigürasyon uzayını test etmek ile aynı sonuca gelecektir.

Eğer bu probleme B-KES yaklaşımı uygulanmış olsaydı, Şekil 2'de verilen 2 li kapsayan dizisine bir tane ekstra konfigürasyon daha eklenerek KK kriteri altında tam bir kapsama elde edilebilirdi. B-KES yöntemi için KK kriteri altında tüm karar sonuçları bir kısıt şeklinde ifade edilmelidir: $p_1, \neg p_1, p_2, \neg p_2, p_3, \neg p_3, p_4, \neg p_4, p_5, \neg p_5, (p_1 \wedge p_2 \wedge p_3), \neg(p_1 \wedge p_2 \wedge p_3), ((p_1 \wedge p_2 \wedge p_3) \wedge (p_4 \vee p_5))$, ve $((p_1 \wedge p_2 \wedge p_3) \wedge \neg(p_4 \vee p_5))$. Şekil 2'de verilen 2 li kapsayan dizisi $((p_1 \wedge p_2 \wedge p_3) \wedge \neg(p_4 \vee p_5))$ dışında tüm ikili parametre ilişkilerini kapsamaktadır. Dolayısıyla, bu karar sonucunun kapsanması için p_1=D, p_2=D, p_3=D, p_4=Y, ve p_5=Y şeklinde bir konfigürasyon daha üretilmelidir.

Örnek 2: Şekil 3a varsayımsal bir yazılım sistemini modelleyen bir sonlu durum makinesini göstermektedir. Modelde başlangıç (s_i) ve bitiş durumlarına ek olarak (s_f) 4 tane durum (s_1, s_2, s_3, s_4) ve 4 tane de sadece iki değer alabilen (doğru için D, yanlış için Y) parametreler bulunmaktadır. Bir parametre sadece 1 tane durum içinde tanım-

lanabilir ve bir durum hiç olmamakla beraber farklı sayılarda parametre içerebilir. Örneğin, Şekil 3a'da durumların hepsi sadece 1 parametre içermektedir. Parametrelere değerler sadece kendi bulundukları durumlarda atanabilir. Parametrenin değeri bir durum içinde bir kez atandıktan sonra, bu parametre bulunduğu durumunda içinde olduğu durumlardan oluşmuş herhangi bir yoldaki tüm parametrelerle etkileşebilir. Örneğin, p_2 parametresine sadece s_2 durumunda değer atanabilir ve bir kez atandıktan sonra p_1 ve p_4 parametreleri ile etkileşebilir (Şekilde 3a'daki $<s_i$-s_1-s_2-s_4-$s_f>$ yolundan ötürü). Durumlar arasındaki geçişler parametrelerin değerleri tarafından korunabilirler. Örneğin, s_1 ile s_2 arasındaki geçiş sadece p_1 doğru olduğunda mümkün olabilir. Ayrıca sistemi başlangıç durumundan bitiş durumuna götüren test parametreleri ve değerleri kümesi test durumu olarak düşünülmüştür. Bu tip bir sistem modellemesi mesela bir telefon uygulaması testinde kullanılabilir. Her durum grafiksel arayüzdeki ekran görüntüsü, parametreler ekrandaki kullanıcı girdileri ve durumlar arasındaki geçişler ise ekranlar arasındaki girdiye bağlı sağlanan geçişleri olarak düşünülebilir.



Şekil. 3. Varsayımsal bir yazılım sistemini modelleyen sonlu bir durum makinesi

Yazılım geliştiricilerin bizim bu varsayımsal sistemimizin tüm ikili parametre değer kombinasyonlarını test etmek istediklerini düşünelim. Geleneksel kapsayan diziler kullanılarak sistemdeki 4 parametre için 2 li bir kapsayan dizi üretilebilir (Şekil 4). Fakat, p_3, p_1=Y ($\neg p_1$) olduğunda erişilemeyeceğinden ötürü, 16 ikili parametre kombinasyonundan 3 tanesi (%18.75) maskelenmiş olacak [14]. Yani bu 3 kombinasyon verilen kapsayan dizi tarafından kapsanmadığı için test edilemeyecekler ve yazılım geliştiricilerinin tüm parametre kombinasyonlarını test ettiklerini düşünmelerine sebep olacaklar.

Maskelenen kombinasyonlardan biri olan (p_3 = D, p_4 = D) Şekil 4'te sadece ilk konfigürasyonda (p_1 = D, p_2 = Y, p_3 = D, p_4 = D) bulunmaktadır. Fakat bu konfigürasyon, p_3'ün p1=D olduğu zamanlarda ulaşılamayacağından dolayı uygulanamayacaktır. Bu sebeple, bu kombinasyon Şekil 4'te verilmiş olan kapsayan dizi tarafından test edilemeyecektir. Diğer maskelenen kombinasyonlar ise şunlardır: (p_2 = Y, p_4 = Y) ve (p_2 = D, p_4 = D).

p_1	p_2	p_3	p_4
D	Y	D	D
D	D	Y	Y
D	Y	Y	Y
Y	D	D	Y
Y	Y	Y	D
Y	D	Y	D

Şekil. 4. Şekil 3a için üretilmiş geleneksel bir 2 li kapsayan dizi.

Başka bir yaklaşım ise bu sistem modelinin her yolu için üzerinde bulunan parametreler kullanılarak kapsayan diziler üretmek olabilir. Daha sonra üretilen bu kapsayan

diziler sistemin test havuzunu oluşturmak için birleştirilebilirler. Örneğin, Şekil 3a'da verilen modelin $<s_i\text{-}s_1\text{-}s_2\text{-}s_4\text{-}s_f>$ ve $<s_i\text{-}s_1\text{-}s_3\text{-}s_4\text{-}s_f>$ yolları için ayrı ayrı 2 li kapsayan diziler oluşturulması gerekir.

Bu alternatif yaklaşımın bir problemi daha karmaşık sistem modelleri için dallanmalar arttıkça olası bütün yolların sayısı da üssel bir hızda artacaktır ve çoğu zaman bütün bu yollar için ayrı ayrı kapsayan diziler oluşturmak mümkün olmayacaktır. Örneğin, Şekil 3b'de başlangıç durumundan bitiş durumuna gidilebilecek 4 farklı yol vardır. Bütün bu yollar için kapsayan diziler üretilmesi gerekir.

Bu yöntemi uygulamak mümkün olsa dahi, her yol için kapsayan dizi üretip daha sonra bunlar birleştirildiğinde, test durumlarının sayısı gerekenden çok fazla bir şekilde artacaktır. Bu da sistemi test etmenin maliyetini arttıracaktır. Bir başka örnek olan Şekil 3c'de ise 2 tane 2 li kapsayan diziye ihtiyaç vardır: bir tanesi $<s_i\text{-}s_1\text{-}s_3\text{-}s_4\text{-}s_5\text{-}s_6\text{-}s_f>$ yolu üzerindeki p_2, p_4, p_5 ve p_6 parametreleri için, diğeri ise $<s_i\text{-}s_1\text{-}s_2\text{-}s_4\text{-}s_5\text{-}s_6\text{-}s_f>$ yolu üzerindeki p_3, p_4, p_5 ve p_6 parametreleri için. Bahsedilen yollarda bulunan p_1'in test durumlarında olmamasının sebebi her yol üzerinde de alacağı değerlerin (D veya Y) sabit olmasındandır. Her iki kapsayan dizinin boyutları en azından 6 olacağından (Şekil 4), birleştirilmiş kapsayan dizinin boyutu ise en az 12 olacaktır. B-KES ise verilen sistem modellemesini kullanarak mümkün olan geçerli tüm parametre değer kombinasyonlarını kısıtlar şeklinde tanımlar. Örneğin, Şekil 3c'de verilen sistem modeli için (p_3=D, p_4=Y) ikilisi $p_2 \wedge \neg p_4$ şeklinde kısıta çevrilebilir.

Geçerli tüm 2 li parametre değerleri kombinasyonlarını gösteren kısıtları kapsamak için Şekil 5'de gösterilen boyutu 8 olan bir B-KES objesi oluşturulabilir. Geleneksel kapsayan dizileri kullandığımız önceki çözümlerimize kıyasla, B-KES 2 li kapsama kriteri ile tam kapsamayı koruyarak test havuzunun boyutunu %33 oranında azaltmıştır.

p_1	p_2	p_3	p_4	p_5	p_6
D	D	*	D	D	Y
D	Y	*	D	Y	D
D	D	*	Y	Y	D
D	Y	*	Y	D	Y
Y	*	Y	Y	Y	Y
Y	*	Y	D	D	D
Y	*	D	Y	Y	Y
Y	*	D	D	D	D

Şekil. 4. Şekil 3c senaryosu için oluşturulmuş bir B-KES objesi. *'lar "önemsiz" durumları göstermektedir. Yani, bu kısımlar geçerli herhangi bir değer ile değiştirildiğinde B-KES objesinin kapsama özelliklerini etkilemezler.

5 B-KES Hesaplama Yöntemi

Bu bölümde B-KES objesi hesaplayabilmek için aç gözlü (greedy) bir algoritma öneriyoruz. Algoritma girdi olarak kısıtlarla tanımlanmış bir sistem modeli *M* ve bir kapsama kirteri *C*, alır. Algoritma ilk olarak tüm geçerli B-KES test isterlerini üretir. Bunu ya-

pabilmek için, öncelikle kapsanması istenen her bir ister üretilir ve bir kısıta r dönüştürülür. Daha sonra tüm r için $r \wedge M$ 'nin tatmin edilebilir olup olmadığı kontrol edilir. Eğer tatmin edilebilirse r geçerli bir isterdir ve test havuzu tarafından kapsanması gerekir. Değilse geçersiz olarak işaretlenir.

Bütün geçerli isterler kümesi R bu şekilde üretildikten sonra, R'nin alt kümelerinden R' oluşmuş başka bir küme S daha oluşturulur, öyle ki; her bir alt küme R' içindeki bulunan kısıtlar kendi bulundukları kümede birlikte tatmin edilebilir olsun. Bu alt kümeler kümesi S oluşturulurken R içindeki her bir r'in mutlaka en azından bir alt kümede bulunduğu garanti edilir. S'in içindeki her bir kümenin daha sonra test durumlarına dönüştürüleceğini göz önünde bulundurarak, içerdiği alt kümeleri sayısı mümkün olduğunca az olması gerektiği söylenebilir.

Bu S kümesini oluşturabilmek için öncelikle boş bir alt küme havuzu ile başlarız. Akabinde, R içindeki bütün geçerli isterleri (r) havuzdaki herhangi bir alt kümeye eklenebilir mi onu kontrol ediyoruz. Eğer böyle bir alt küme bulunursa, r'yi bu alt kümeye ekleriz. Eğer bulunamazsa, havuza yeni boş bir alt küme ekler ve r'yi bu yeni kümeye koyarız. Burada dikkat edilmesi gereken husus alt kümelerde bulunan tüm isterler kısıtların birleşimi şeklinde ifade edilmiştir $\bigwedge_{r' \in R'} r'$. Bundan dolayı, bir isterin r var olan bir alt kümeye eklenip eklenememesi, aslında bu alt kümedeki kısıtların r ve M ile birlikte tatmin edilebilir olup olmamasıdır: yani $r \wedge M \wedge (\bigwedge_{r' \in R'} r')$. Sonuç olarak, eğer bu kısıt ifadesinin sonucu tatmin edilebilir ise, r bu alt kümeye eklenir.

Bütün alt kümeleri oluşturduktan sonra, tüm alt kümeler R′ için, $r \wedge M \wedge (\bigwedge_{r' \in R'} r')$. ifadesi çözülerek her bir alt küme için farklı bir test durumu oluşturuyoruz. Bu sayede oluşturulmuş bu test durumlarının verilen kapsama kriteri C altında tam kapsama yaptığı garanti edilmiş oluyor.

6 Sonuç ve Gelecek Çalışmalar

Bu çalışmamızda sunmuş olduğumuz yaklaşımımızın, KES yaklaşımlarının pratik hayattaki kullanabilirliklerini önemli ölçüde arttıracağını düşünüyoruz. Bu yüzden, B-KES objeleri üretebilmek adına yeni diller ve geleneksel olmayan tipte konfigürasyon uzaylarını ve kapsama krtierlerini tanımlayabilmek için yeni araçlar ve algoritmalar geliştirmeye devam edeceğiz.

7 Kaynaklar

1. Hartman, A.: Software and hardware testing using combinatorial covering suites. In Graph theory, combinatorics and algorithms pp. 237-266. Springer US (2005).
2. Yilmaz, C., Fouche, S., Cohen, M.B., Porter, A., Demiroz, G. and Koc, U.: Moving forward with combinatorial interaction testing. Computer, (2), pp.37-45 (2014).
3. Williams, A.W. and Probert, R.L.: Formulation of the interaction test coverage problem as an integer program. In Testing of Communicating Systems XIV pp. 283-298. Springer US (2002).
4. Harman, M.: The current state and future of search based software engineering. In 2007 Future of Software Engineering pp. 342-357. IEEE Computer Society (2007).

5. Schroeder, P.J., Faherty, P. and Korel, B.: Generating expected results for automated black-box testing. In Automated Software Engineering, Proceedings. 17th IEEE International Conference on pp. 139-148. IEEE (2002).
6. Kuhn, R., Lei, Y. and Kacker, R.: Practical combinatorial testing: Beyond pairwise. IT Professional, 10(3), pp.19-23 (2008).
7. Yilmaz, C., Cohen, M.B. and Porter, A.A.: Covering arrays for efficient fault characterization in complex configuration spaces. Software Engineering, IEEE Transactions on, 32(1), pp.20-34 (2006).
8. Mercan, H. and Yilmaz, C.: Pinpointing Failure Inducing Event Orderings. In Software Reliability Engineering Workshops (ISSREW), IEEE International Symposium on pp. 232-237. IEEE (2014).
9. Johansen, M.F., Haugen, Ø. and Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. InProceedings of the 16th International Software Product Line Conference-Volume 1 pp. 46-55. ACM (2012).
10. Cohen, M.B., Gibbons, P.B., Mugridge, W.B. and Colbourn, C.J.: May. Constructing test suites for interaction testing. In Software Engineering. Proceedings. 25th International Conference on (pp. 38-48). IEEE (2003).
11. Yuan, X., Cohen, M.B. and Memon, A.M.: GUI interaction testing: Incorporating event context. Software Engineering, IEEE Transactions on,37(4), pp.559-574,Vancouver (2011).
12. Mercan, H., Yilmaz C.: A Constraint Solving Problem Towards Unified Combinatorial Interaction Testing, accepted for publication in the 2016 Proceedings of the International workshop on Constraints in Software Testing, Verification and Analysis (CSTVA'16), Saarbruecken, Germany, (2016).
13. Nie, C. and Leung, H.: A survey of combinatorial testing. ACM Computing Surveys (CSUR), 43(2), p.11 (2011).
14. Dumlu, E., Yilmaz, C., Cohen, M.B. and Porter, A.: Feedback driven adaptive combinatorial testing. In Proceedings of the International Symposium on Software Testing and Analysis pp. 243-253. ACM (2011).
15. Kobayashi, N.: Design and evaluation of automatic test generation strategies for functional testing of software. Osaka, Japan, Osaka Univ (2002).
16. Bryce, R.C. and Colbourn, C.J.: One-test-at-a-time heuristic search for interaction test suites. In Proceedings of the 9th annual conference on Genetic and evolutionary computation pp. 1082-1089. ACM (2007).
17. Cohen, M.B., Colbourn, C.J. and Ling, A.C.: Augmenting simulated annealing to build interaction test suites. In Software Reliability Engineering. 14th International Symposium on pp. 394-405. IEEE (2003).
18. Yilmaz, C.: Test case-aware combinatorial interaction testing. Software Engineering, IEEE Transactions on, 39(5), pp.684-706 (2013).
19. Elbaum, S., Malishevsky, A.G. and Rothermel, G.: Test case prioritization: A family of empirical studies. Software Engineering, IEEE Transactions on, 28(2), pp.159-182 (2002).
20. Bryce, R.C. and Colbourn, C.J.: Prioritized interaction testing for pair-wise coverage with seeding and constraints. Information and Software Technology, 48(10), pp.960-970 (2006).
21. Qu, X., Cohen, M.B. and Woolf, K.M.: Combinatorial interaction regression testing: A study of test case generation and prioritization. In Software Maintenance. IEEE International Conference on pp. 255-264. IEEE (2007).
22. Srikanth, H., Cohen, M.B. and Qu, X.: Reducing field failures in system configurable software: Cost-based prioritization. In 20th International Symposium on Software Reliability Engineering pp. 61-70. IEEE (2009).
23. Kimoto, S., Tsuchiya, T. and Kikuno, T.: Pairwise testing in the presence of configuration change cost. In Secure System Integration and Reliability Improvement. Second International Conference on pp. 32-38. IEEE (2008).