Olay Tabanlı Sınama İçin Mutant Seçimi

Fevzi Belli, Mutlu Beyazıt

EIM-E/ADT, Paderborn Üniversitesi, Paderborn, Almanya
{belli, beyazit}@adt.upb.de

Özet. Model tabanlı sınama, sınama örneği üretimi için biçimsel modeller kullanmayı içerir. Bu bildiri olay tabanlı modelleme için *düzenli gramerler*i önermektedir. Önerilen modeli değişikliklere uğratmak için tanımlanan *mutasyon işleçleri* sistematik olarak *hata modelleri* ya da *mutant*ların üretilmesinde kullanılmaktadır. Asıl sistem modeliyle mutantlar üzerinde uygulanan algoritmalar ile sınama örnekleri üretilmektedir. Mevcut yöntemler birer olaya odaklanırken bu bildirideki yaklaşım $k \geq 1$ uzunluğundaki olay ardışımlarına (*k-ardışımları*na) odaklanmakta ve aşamalı olarak farklı hataların modellenmesini sağlamaktadır. Yaklaşım ayrıca mutasyon tabanlı sınamada karşılaşılan şimdiye kadar çözülmemiş şu sorunlar ile de başa çıkmaktadır: (i) Asıl modele denk mutantların ve (ii) aynı hatayı modelleyen birden fazla mutantın ortadan kaldırılması. Bu tür mutantlar kaynakların israfına ve sınama sürecinde verim kaybına yol açmaktadır. Önerilen yaklaşım çerçevesinde söz konusu mutantların üretilmeye bile gereksinim duyulmadan dışlanması için mutant seçme taktikleri geliştirilmektedir. Ayrıca, yaklaşımın bir örnek çalışma üzerinde varolan olay ardışım çizgeleri tabanlı yaklaşımla kıyaslanarak geçerliliği gösterilmektedir.

Anahtar Kelimeler. model tabanlı mutasyonla sınama, olay tabanlı, mutant seçimi

1 Giriş ve İlgili Çalışmalar

Sınama, sınama girdileri ve *beklenen sınama çıktıları*ndan oluşan *sınama örnekleri*nin kullanımına dayalı kalite güvencesi için kullanılan bir tekniktir. Sınama esnasında sınama örnekleri kümesi (*sınama kümesi*) *sınama altındaki sistem* üzerinde işletilir. Sınama girdilerine karşılık gelen beklenen çıktıları üretilirse sistem sınamayı *geçer*; aksi takdirde sınamadan *kalır*. Bu beraberinde gözlenen ve beklenen çıktıların örtüşüp örtüşmediğine karar vermeyi gerektiren *kehanet (oracle) sorunu*nu da getirir. *Kapsama kriteri* [20] sınama kümesinin kalitesinin bir ölçüsüdür ve sınamayı için durdurma koşulu olarak kullanılır.

Model tabanlı sınama sistemin *model* adı verilen bir soyutlamasının oluşturulmasına ve bu modelin sistemin sınanmasında kullanılmasına dayanmaktadır [4]. Modellerin kullanımının birçok getirisi vardır. Örnek olarak, kullanılan model beklenen çıktıların belirlenmesi için uygunsa kehanet sorunundan kaçınarak hata tespiti ve maliyet açısından etkinlik ve verimlilik artışı sağlanabilir [19].

Model tabanlı mutasyonla sınama [11][2][10] sınama örnekleri üretimi sırasında

ek olarak *hata modeller*inin de kullanımını içerir. Bir hata modeli *mutant* olarak da adlandırılır; çünkü bir *mutasyon işleci* kullanılarak asıl modelden üretilir. Bazı mutantlar asıl modele denk olabilir. Bu önemli bir sorundur; çünkü *denk mutantlar* herhangi farklı bir davranış tanımlamaz ve sınama kaynaklarının israfına neden olur. Örnek olarak, Grün, Schuler ve Zeller [12] üretilen mutantların %40'ına kadarının denk olabileceğini göstermiştir. Model tabanlı mutasyonla sınama mutantlar kullanarak bu mutantları asıl modelden ayırt eden (*öldüren*) sınama örnekleri üretmeyi amaçlamaktadır. Böyle bir sınama örneği sistemde işletildiğinde mutant tarafından modellenen hatanın sistemde var olup olmadığı sınanabilir.

Doğrudan sistem modeli kullanılarak elde edilen sınama örnekleri genelde *olumlu sınama*da kullanılır; sistemin kendisinde beklenenleri yerine getirmesi sınanır. *Olumsuz sınama* da ise belli tip mutantlar kullanılır ve sistemin kendisinden beklenmeyenleri yerine getirmemesi sınanır.

Gramer tabanlı sınama da ilgi gören bir tekniktir [17]. Çoğunlukla gramerler *"grammarware"* adı verilen derleyici, hata ayıklayıcı, kod üreteci ve belge üreteci gibi yazılımların sınanmasında kullanılmaktadır [16]. Bu çerçevedeki yaklaşımlarda genelde bağlam bağımsız gramerler kullanılarak programlar için girdi üretilmektedir.

Bu bildiride ele alınan yaklaşım model tabanlı mutasyonla sınama için geliştirilmiş olay tabanlı yaklaşımlardan [10][6][7][14] birçok yönden farklıdır. Yaklaşım modelleme için $k \geq 1$ uzunluktaki olay ardışımlarını (*k-ardışımları*nı) ele alan bir olay tabanlı düzenli gramer modeli kullanmaktadır. Hata modelleri üretmek için ilgili mutasyon işleçleri tanımlanmakta ve k değerinin farklılaşmasıyla aşamalı olarak farklı hataların modellenebilmesi sağlanmaktadır. Ayrıca, yaklaşım asıl modele denk olan mutantların ve aynı hatayı modelleyen birden fazla mutantın dışlanması için mutant seçme taktikleri içermektedir. Bu sayede denklik denetimi için asıl modelle mutantın kıyaslanmasını gerektiren diğer model tabanlı mutasyonla sınama yaklaşımlarının (mesela [2][3]) aksine belirtilen mutantlar daha üretilmeden dışlanmaktadır. Ek olarak, aynı hataları modelleyen birden fazla mutant da dışlanmaktadır.

Kısım 2'de olay tabanlı gramer modeli ve ilişkin kavramlar örneklerle sunulmaktadır. Ardından, Kısım 3 sınama kümesi üretmek için mutant seçme taktiklerini tartışmaktadır. Öne sürülen yaklaşımın daha önce öne sürülmüş olan olay ardışım çizgesi tabanlı yaklaşımla kıyaslanarak bir örnek çalışma üzerinde değerlendirilmesi Kısım 4'te yapılmakta ve Kısım 5 ile bildiri sonlandırılmaktadır.

2 Olay Tabanlı Sınama için Gramerler

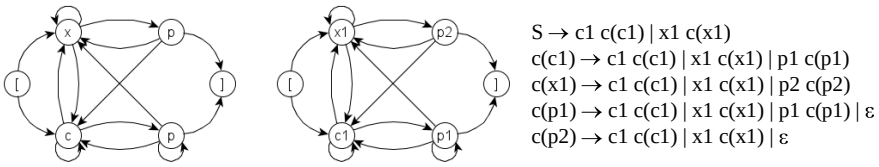
Genel olarak, bir *olay* çevresel bir uyarı veya sistem tepkisi gibi sistem etkinliğinin farklı aşamalarında meydana gelen ve dışarıdan gözlemlenebilen bir olgudur. Model tabanlı sınama sistemin iç içleyişine ait detayı göz ardı eder, ve böylece göreceli olarak daha soyut ve basit betimlemeler kullanır [9]. Bu bildiride öne sürülen yaklaşımın öğeleri olaylar olan biçimsel bir gramer modelini kullanmasının neden budur; olaylar sınayıcı tarafından gözlemlenebilir olduğundan sistemin sınamayı geçip geçmediğine karar verilebilmesini sağlarlar (Kehanet sorunu; Kısım 1'e bakınız).

Örnek 1. Diyelim ki "kes-kopyala-yapıştırma" işlemindeki üç olay var:

c: kopyalama, *x*: kesme ve *p*: yapıştırma.

Başlangıçta, *c* veya *x* gerçekleşebilir. *c* olayı *c*, *x* veya *p*'nin biri tarafından takip edilebilir; aynı durum *x* olayı için de geçerlidir. *p* olayı *c*'den sonra meydana gelirse, *c*, *x* veya *p*'nin biri tarafından izlenebilir. Fakat, *p* olayı *x*'ten sonra gerçekleşirse, ya *c* ya da *x* tarafından takip edilebilir; diğer bir deyişle, bir nesneyi kesip yapıştırdıktan sonra tekrar yapıştırmak mümkün değildir. Bir *p* olayından sonra etkinlik sonlanabilir.

Şekil 1a Örnek 1 için olay tabanlı bir yönlü çizge modelidir. Bu tür modeller sınamada sıkça kullanılmaktadır [9] ve sonlu durum devinirleri (çıktıların dahil edilmesi durumunda çıktılı sonlu durum devinirleri) ile aynı anlatım gücüne sahiptir. Olaylar dışarıdan gözlemlenebilir olguları ve durumlar sistemin iç işleyişine ait olguları temsil ettiklerinden öne sürülen modelde olaylar üzerine odaklanılmakta ve durumlar görselleştirilmemektedir. Bu sebeple olaylar düğümlere yerleştirilmekte ve olaylar arasındaki *izleme bağıntısı* kenarlar kullanılarak tanımlanmaktadır. *[* ve *]* sahte olayları başlangıç ve bitiş olaylarını işaretlemek için kullanılmaktadır. [5]



(a) Belirsizlik içeren model. (b) Bağlamlı olaylar içeren model (Belirsizlik yok). (c) 1-ardışımlarını kullanan gramer modeli.

Şekil 1. Örnek 1 için olay tabanlı modeller.

Şekil 1a'daki modelin ciddi bir sorunu vardır: *p* olayından söz edilmek istendiğinde, hangi *p* olayından bahsedildiği net olarak anlaşılmamaktadır. Bu tür belirsizliklerden kaçınmak için bu bildiride öne sürülen model olayları içinde bulundukları *bağlamlar*a göre farklılaştırmakta ve *bağlamlı olaylar* kullanmaktadır; Şekil 1b'de gösterilen *{c1, x1, p1, p2}* gibi. Aynı zamanda olaylar kullanıcı tarafından görülen haliyle de saklanmaktadır; *{c, x, p}*. Bu olaylara *köken olaylar* denilmektedir.

Ayrıca, bu tür bir model sadece tek olaylar arasındaki ilişkileri inceler. Olayları *k≥1* uzunluktaki olay ardışımlarına (*k-ardışımları*na) göre inceleyen bir olay tabanlı model tanımlamak için gramerlerin [15] kullanılması uygundur; çünkü bir gramer modeli birden fazla olayın *kuralları* içerisinde yer almasına izin vermektedir.

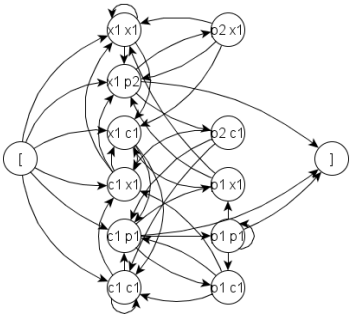
Şekil 1c'de Örnek 1'i 1-ardışımları kullanarak betimleyen bir gramer modelinin kuralları verilmektedir; *c(r), r ardışımının bağlamı*nı temsil etmektedir. Ayrıca, bu kuralların yönlü çizge kullanılarak görselleştirilmesi de mümkündür (Şekil 1b).

Örnek 1'i 2-ardışımları ile modellemek için kuralları Şekil 2'de verilen gramer kullanılabilir. Şekil 1c ve Şekil 2'deki gramerler aşağıdaki kural anlamları kullanıldığında aynı sistemi betimler.

- Kural $c(a\ e) \rightarrow e\ b\ c(e\ b)$ (veya kenar *(a e, e b)*): *b*, *a*'yı izler.
- Kural $S \rightarrow a\ b\ c(a\ b)$ (veya kenar *([, a b)*): *a b* bir başlangıç 2-ardışımıdır.
- Kural $c(a\ b) \rightarrow \varepsilon$ (veya kenar *(a b,])*): *a b* bir bitiş 2-ardışımıdır.

S → c1 c1 c(c1 c1) | c1 x1 c(c1 x1) | c1 p1 c(c1 p1) |
x1 c1 c(x1 c1) | x1 x1 c(x1 x1) | x1 p2 c(x1 p2)
c(c1 c1) → c1 c1 c(c1 c1) | c1 x1 c(c1 x1) | c1 p1 c(c1 p1)
c(c1 x1) → x1 c1 c(x1 c1) | x1 x1 c(x1 x1) | x1 p2 c(x1 p2)
c(c1 p1) → p1 c1 c(p1 c1) | p1 x1 c(p1 x1) | p1 p1 c(p1 p1) | ε
c(x1 c1) → c1 c1 c(c1 c1) | c1 x1 c(c1 x1) | c1 p1 c(c1 p1)
c(x1 x1) → x1 c1 c(x1 c1) | x1 x1 c(x1 x1) | x1 p2 c(x1 p2)
c(x1 p2) → p2 c1 c(p2 c1) | p2 x1 c(p2 x1) | ε
c(p1 c1) → c1 c1 c(c1 c1) | c1 x1 c(c1 x1) | c1 p1 c(c1 p1)
c(p1 x1) → x1 c1 c(x1 c1) | x1 x1 c(x1 x1) | x1 p2 c(x1 p2)
c(p1 p1) → p1 c1 c(p1 c1) | p1 x1 c(p1 x1) | p1 p1 c(p1 p1) | ε
c(p2 c1) → c1 c1 c(c1 c1) | c1 x1 c(c1 x1) | c1 p1 c(c1 p1)
c(p2 x1) → x1 c1 c(x1 c1) | x1 x1 c(x1 x1) | x1 p2 c(x1 p2)

(a) Kurallar.



(b) Yönlü çizge hali.

Şekil 2. Örnek 1 için 2-ardışımlarını kullanan bir gramer modeli.

Yukarıda verilen sezgisel bilgiler ışığında k-ardışımlarını kullanan olay tabanlı gramer modelinin tanımı aşağıda verilmektedir.

Tanım 1 (k-ardışım Sağ Düzenli Gramer (*k-DG*)). *Bir k-ardışım sağ düzenli grameri (k-DG) (tam sayı $k \geq 1$) bir altılıdır: $G = (E, B; K; C; S; P)$. Burada:*

- *E, sonlu bir olaylar (bağlamlı olaylar) kümesidir.*
- *B, sonlu bir köken olaylar kümesidir. Her $e \in E$ için, $d(e) \in B$ karşılık gelen köken olaydır, ve d(.) bağlamdan arındırma fonksiyonudur.*
- *$K \subseteq E^k$, sonlu bir k-ardışımlar (ya da uç simgeler) kümesidir. Her $r \in K$ için, $r = r_1 \ldots r_k$; ve $d(r) = d(r_1) \ldots d(r_k) \in B^k$, r'ye karşılık gelen köken olay k-ardışımıdır.*
- *C, sonlu bir bağlamlar (ya da ara simgeler) kümesidir.*
- *S, başlangıç bağlamıdır (ya da başlangıç simgesidir).*
- *P, sonlu bir kurallar kümsedir. Her bir kural aşağıdaki biçimleden birine sahiptir:*

$$Q \rightarrow \varepsilon \text{ veya } Q \rightarrow r\ c(r).$$

Burada $Q \in C$, bir bağlam; $r \in K$, bir k-ardışımı; $c(r) \in C \backslash \{S\}$; r'nin eşsiz bağlamı; ve ε, boş dizgidir. $k \geq 2$ ise, her $c(q) \rightarrow r\ c(r) \in P$ için

$$q_2 \ldots q_k = r_1 \ldots r_{k-1}.$$

Aksi belirtilmediği taktirde, bildirini geri kalanında *G*'nin *(E, B; K; C; S; P)* biçiminde herhangi bir k-DG olduğu varsayılmaktadır.

Kuralların anlamları izleyen şekildedir: Her $c(q) \rightarrow r\ c(r) \in P$ için r_k *olayı q k-ardışımını gramer G tarafından modellenen sistemde izler; yani, q* r_k bu sistemde bir (k+1)-ardışımıdır. Ayrıca, her $S \rightarrow r\ c(r) \in P$ için, *r* bir *başlangıç k-ardışımı*; ve her $c(q) \rightarrow \varepsilon \in P$ için, *q* bir *bitiş k-ardışımı*dır.

k-DG kuralları dizgiler türetmek için kullanılabilir. Bir *türetme*, türetme adımlarının ardışımlarından oluşur ve $\Rightarrow_G^*$ ile gösterilir; her bir *türetme adımı* ise $xQy \Rightarrow_G xRy$ ($x, y \in (E \cup N)^*$ ve $Q \rightarrow R \in P$) şeklindedir (Karmaşıklık oluşmadığı durumlarda göreceli olarak $\Rightarrow^*$ ve $\Rightarrow$ kullanılabilir). Gramer *G* tarafından betimlenen dil $L(G) = \{w \in E^* \mid S \Rightarrow^* w\}$ ile ifade edilir.

Bir k-DG, kendisine karşıık gelen (k+1)-DG'ye algoritmik olarak dönüştürülebilir [6][7]. Bu sayede verien bir sistem için 1-DG modeli tasarlandıktan sonra bu sistem

için herhangi bir k-DG modeli otomatik olarak elde edilebilir.

k-DG kuralları yönlü çizgeler ile görselleştirilebilir. Düğümleri etiketlemek için k-ardışımları ve *[* ile *]* kullanılır. "*([, r)*", "*(r,])*" ve "*(q, r)*" biçimdeki kenarlar sırasıyla "$S \rightarrow r\ c(r)$", "$c(r) \rightarrow \varepsilon$" ve "$c(q) \rightarrow r\ c(r)$" biçimindeki kurallara karşılık gelir. Bu tür yönlü çizgelerden anlatımı kolaylaştırmak için sıkça faydalanılmaktadır.

Tanım 1'de verilen *d(.)* fonksiyonu bağlamlı ardışımları köken ardışımlarla ilişkilendirmek için aşağıda genişletilmektedir.

Tanım 2 (Bağlamdan Arındırma Fonksiyonu). *$s = s_1\ s_2\ \ldots \in E^*$ bir olay ardışımı ve X bir olay ardışımları kümesi olsun.Eğer $s \neq \varepsilon$ ise, s'ye karşılık gelen köken olay ardışımı $d(s) = d(s_1)\ d(s_2)\ \ldots \in B^*$; eğer $s = \varepsilon$, $d(\varepsilon) = \varepsilon$ olarak tanımlanır. X'e karşılık gelen köken olay ardışımları kümesi $d(X) = \{d(s) |\ s \in X\}$'dir.*

Örnek 2 (Bağlamdan Arınmış Olay Ardışımları). Şekil 1c'deki 1-DG düşünüldüğünde, *X = {c1, c1 p1, c1 x1 p2}* için, *d(X) = {c, c p, c x p}* olmaktadır.

Sınama için k-DG kuralları kullanılarak elde edilebilecek ve elde edilemeyecek olay ardışımları birbirinden farklılaştırılmaktadır.

Tanım 3 (k-DG İçindeki Olay Ardışımları). *Eğer $Q \Rightarrow^* xsy$ ($Q \in C$ ve $x,y \in (C \cup Æ)^*$) biçiminde bir türetme varsa, s olay ardışımı G grameri içindedir. Eğer $S \Rightarrow^* s\ Q$ ($Q \in C$) [ya da $Q \Rightarrow^* s$ ($Q \in C$)] biçiminde bir türetme varsa, boş olamayan s olay ardışımı bir başlangıç [ya da bitiş] ardışımıdır. G gramerinde olmayan bir olay ardışımına hatalı olay ardışımı da denir.*

Örnek 3 (1-DG İçindeki Olay Ardışımları). Şekil 1c'deki 1-DG için:

- *{c1 x1, x1 p2, p1 p1}* kümesindeki 2-ardışımları gramerin içindedir; *{p2 p1, p2 p2}* kümesindekiler ise değildir.
- *{c1, x1, c1 c1, x1 x1 p2, c1 p1 x1}* kümesi bir başlangıç ardışımları kümesidir ve *{p1, p2, p1 p1, x1 p2}* kümesi bir bitiş ardışımları kümesidir.

k-DG'ler için sınama örnekleri aşağıdaki şekilde tanımanmaktadır.

Tanım 4 (Sınama Örnekleri).

- *Bir olay ardışımı s G grameri içinde bir başlangıç ardışımı veya $s=\varepsilon$ ise s bir olumlu sınama örneğidir. $T_P(G)$ tüm olumlu sınama örnekleri kümesini ifade etmektedir. Bir olumlu sınama örneği s G grameri içinde hem başlagıç hem de bitiş ardışımı veya $\varepsilon \in L(G)$ iken $s=\varepsilon$ ise, s bir tam olay ardışımıdır. $T_{CES}(G) = L(G) \subseteq T_P(G)$ tüm tam olay ardışımlarının kümesini temsil eder.*
- *Bir olay ardışımı s'nin ilk olayı başlangıç olayı değilse veya s'de bulunan 2-ardışımlarından en azından biri G grameri içinde değilse s bir olumsuz sınama örneğidir. $T_N(G)$ tüm olumsuz sınama örnekleri kümesini ifade etmektedir. Bir olumsuz sınama örneği s sadece başlangıç olayı olamayan bir olaydan oluşuyorsa veya G grameri içinde olmayan sadece bir adet 2-ardışımı içeriyor ve bu ardışımla sonlanıyorsa s bir hatalı tam olay ardışımıdır. $T_{FCES}(G) \subseteq T_N(G)$ tüm hatalı tam olay ardışımlarının kümesini temsil eder.*
- *Sınama örnekleri kümesine sınama kümesi denir.*

Örnek 4 (1-DG'nin Sınama Örnekleri). Şekil 1c'deki 1-DG için:

- *{c1, x1 x1, c1 p1 p1 x1}* bir olumlu sınama kümesidir, ve *{x1 p2, x1 x1 p2, c1 p1 p1*

p1} bir tam olay ardışımları kümesidir.

- *{p1, x1 p2 p1 c1, c1 x1 p2 p2}* bir olumsuz sınama kümesidir, ve *{x1 p2 p2, c1 x1 p2 p2}* bir hatalı tam olay ardışımları kümesidir.

k-DG olayları bağlamlıdır. Fakat, sistem davranışları köken olaylar ile ifade edilir; çünkü köken olayları, olayların kullanıcı tarafından bilinen halini temsil eder (Tanım 1). Bu sebeple k-DG'lerin denkliği izleyen şekilde tanımlanmaktadır.

Tanım 5 (Denklik). *G ve H iki k-DG olsun.* $d(T_{CES}(G)) = d(T_{CES}(H))$ *ise G ve H denktir.*

k-DG içindeki tüm k-ardışımlarının kullanışlılığı uygulamada önem taşımaktadır.

Tanım 6 (Kullanışlılık). *Dizgi* $z \in (C \cup E)^*$ *verilsin. Eğer* $S \Rightarrow^* xzy \Rightarrow^* w$ $(x,y \in (C \cup E)^*$ *ve* $w \in E^*)$ *şeklinde bir türetme varsa z dizgisi G grameri içinde kullanışlıdır. Eğer G gramerindeki tüm k-ardışımları kullanışlı ise G kullanışlıdır.*

Örnek 5 (Kullanışlı ve Kullanışsız k-DG'ler). Şekil 1c ve Şekil 2'deki gramerlerin hepsi kullanışlıdır. Şekil 1c'deki gramerden $c(p1) \rightarrow \varepsilon$ ve $c(p2) \rightarrow \varepsilon$ çıkartırlırsa, kullanışsız bir gramer elde edilir. Bu gramerde hiç bir bitiş olayı yoktur ve $T_{CES}(G)$ boş kümedir. Yine de geriye kalan kurallar olaylar arasındaki izleme ilişkisini doğru şekilde göstermektedir.

Deterministik modeller lüzumsuz olay ardışımlarını dışlamaya yardımcı olur.

Tanım 7 (Determinizm). *G gramerinde her* $Q \in C$ *için* $Q \rightarrow q\ c(q) \in P$ *ve* $Q \rightarrow r\ c(r) \in P$ *öyle ki* $r \neq q$ *ve* $d(r) = d(q)$ *olacak şekilde iki kural yoksa G deterministiktir.*

Örnek 6 (Deterministik 1-DG'nin Sınama Örnekleri). Şekil 1c'ye $c(c1) \rightarrow p2\ c(p2)$ kuralını ekleyerek elde edilen 1-DG deterministik değildir. Bu gramerdeki olumlu sınama örnekleri $s = c1\ c1\ p1$ ve $t = c1\ c1\ p2$ 'den biri lüzumsuzdur; çünkü $d(s) = d(t) = c\ c\ p$ olmaktadır.

Bu bildiride, aksi belirtilmediği takdirde, sözü geçen tüm grameler kullanışlı ve deterministik k-DG'lerdir.

3 Sınama Kümesi Üretmek için Mutant Seçimi

Olay tabanlı hata tipleri *eksik olay* ve *fazla olay* olarak sınıflandırılabilir. Eksik olay hatalarında, bir olay belli bir olay ardışımından önce veya sonra meydana gelemez; fazla olay hatalarında ise bir olayın belli bir olay ardışımından sonra meydana gelmesi sözkonusudur. Bu tip hataları modellemek için daha önce tanımlanmış olay tabanlı mutasyon işleçlerini [10][14] doğru şekilde genişleterek *işaretleme* (*başlangıç işaretle, bitiş işaretle, başlangıç olmayan işaretle* ve *bitiş olmayan işaretle*), *ekleme* (*ardışım ekle* ve *uç simge ekle*) ve *çıkarma* işleçleri (*ardışım çıkar* ve *uç simge çıkar*) tanımlamak mümkündür. Bu işleçler küçük değişiklikler veya modelin belli kısımlarında yerel değişiklikler yapılmasına izin verdiğinden farklı mutantların fazla sayıda ortak hatayı modellemesi ve modellenen bir hatanın diğer bir hataya müdahele etmesi büyük ölçüde önlenebilir. (Aşağıda belirtildiği üzere, yaklaşım sınama örneği üretimi için tüm işleçlerin kullanımına gereksinim duymadığından bu işleçlerden sadece ge-

rekenlerin tanımları ilerleyen kısımlarda verilmektedir.)

Bu bildirideki yaklaşım seçilen mutantların asıl model ile birlikte sınama örneği üretiminde kullanılmasını amaçlamaktadır. Bu yüzden aşağıda verilen ve olay tabanlı sınama için geçerli olan varsayımlar doğrultusunda sözedilen işleçlerden sadece bazıları tanımlanıp kullanılmaktadır (Olay tabanlı sınama ile ilgili daha fazla bilgi için [9] ve oradaki kaynakçaya bakınız).

A1. Sınama örneğindeki olaylar verilen sırada işletilir; sınama örneğinin işletilmesi bir aksaklıkla karşılaşılaca durdurulur.

A2. Bir sınama örneği herhangi bir olayla sonlanabilir; bu olay bitiş olayı olmak zorunda değildir.

Bu durumda aşağıdakiler söylenebilir.

P1. Eksik ve fazla olay hatalarının ele alınmasında sadece eksik veya fazla olan olaydan önce gelen ardışımlar dikkate alınabilir ve bu olayı izleyen ardışımlar görmezden gelinebilir. Bu sayede, k-DG içindeki tüm (k+1)-ardışımları işleterek bir olayın kendisinden önce gelen bir k-ardışımından sonra eksik olup olmadığı anlaşılabilir. Ayrıca, tüm hatalı (k+1)-ardışımları işleterek bir olayın kendisinden önce gelen bir k-ardışımından sonra fazla olup olmadığı bulunabilir. (A1 gereği)

P2. Başlangıç olmayan işaretle, bitiş olmayan işaretle, ardışım çıkar ve olay çıkar mutantlarının kullanılmasına gerek yoktur; çünkü bu mutantlar asıl modelde bulunmayan herhangi bir (k+1)-ardışımı içermemektedir. (P1 gereği)

P3. Bitiş işaretle ve bitiş olmayan işaretle mutantları gerçekten hata modelleri temsil etmemektedirler; çünkü sınama sürecinde hehangi bir olay bitiş olayı veya bitiş olmayan olay olarak düşünülebilir. (A2 gereği)

P4. Ardışım ekle mutantlarıyla modellenen fazla olay hatalarının hepsi uç simge ekle mutantları tarafından modellenebilir. (Tanım gereği [10])

P5. Kullanılan tüm olumsuz sınama örnekleri hatalı tam olay ardışımlarıdır. (A1 gereği)

Sonuç olarak, sınama örneği üretimi için tüm mutant tiplerinin kullanılmasına gerek yoktur. Asıl model kullanılarak eksik olay hatalarının tespiti için (k+1)-ardışımları üretilebilir, ve başlagıç işaretle ve uç simge ekle mutantları kullanılarak fazla olay hatalarının tespiti için hatalı (k+1)-ardışımları üretilebilir. Bu yüzden izleyen Kısım 3.1 ve Kısım 3.2'de başlangıç işaretle ve uç simge ekle işleçleri ele alınarak ilgili mutant seçme taktikleri ortaya konumaktadır.

Bu kısımlarda, aksi belirtilmediği takdirde, $G = (E, B; K; C; S; P)$ asıl k-DG modeli ve G' duruma göre başlangıç işaretle mutantı veya uç simge ekle mutantı olarak düşünülmektedir.

3.1 Başlangıç İşaretle Mutantları

Başlangıç işaretle işleci kullanılarak k-ardışımları başlangıç olarak işaretlenebilir ve mutantlar fazla başlangıç ardışımı hatalarını modellemede kullanılabilir.

Tanım 8 (Başlangıç İşaretle (bİ)). *Verilen bir* $e \in K$ *öyle ki* $S \rightarrow e\ c(e) \notin P$ *için* $G' = bİ(G, e) = (E, B; K; C; S; P \cup \{S \rightarrow e\ c(e)\})$ *G'nin bir başlangıç işaretle (bİ) mutantıdır.*

Örnek 7 (Bir bİ Mutantı). Şekil 1c'deki gramere *G* denilirse Şekil 3a'daki gramer *bİ(G, p1)* mutantıdır.

(a) Başlangıç işaretle (b) Uç simge ekle

Şekil 3. Şekil 1c'deki 1-DG'nin mutantları (Mutasyonlar **kalın** olarak gösterilmiştir).

Mutasyon sonucu tüm tam olay ardışımları kümesi değişmektedir.

Lemma 1 (Bir bİ Mutantının Tam Olay Ardışımları Kümesi). $T_{CES}(G') = T_{CES}(G) \cup \{e\ x |\ c(e) \Rightarrow_G^* x\ (x \in E^*)\}$.
İspat: İspat Tanım 8'den çıkmaktadır. □

Bir başlangıç işaretle mutantının asıl k-DG'ye denkliği aşağıdaki şekildedir.

Lemma 2 (Bir bİ Mutantının Denkliği). *G', G'ye denk değildir ancak ve ancak* $d(X) \setminus d(Y) \neq \varnothing$ *öyle ki*

- $X = \{e\ x |\ c(e) \Rightarrow_G^* x\ (x \in E^*)\}$ *ve*
- $Y = \{e'\ y |\ S \Rightarrow_G^* e'\ y\ (e' \in K, y \in E^*)$ *öyle ki* $e' \neq e$ *ve* $d(e') = d(e)\} \subseteq T_{CES}(G)$.

İspat: $T_{CES}(G') = T_{CES}(G) \cup X$ (Lemma 1). Tanım 5 gereği $d(T_{CES}(G')) = d(T_{CES}(G))$ ancak ve ancak $d(X) \subseteq d(Y) \subseteq d(T_{CES}(G))$. □

Bir başlangıç işaretle mutantının kullanışlılığı, deterministliği ve denk olmaması için yeter koşullar aşağıda verilmektedir.

Teorem 1 (Bir bİ Mutantının Kullanışlılığı). *G kullanışlı ise G' kullanışlıdır.*
İspat: İspat Tanım 6 ve Tanım 8 kullanılarak yapılabilir. □

Teorem 2 (Bir bİ Mutantının Determinizmi). *G deterministik ise ve* $S \rightarrow e'\ c(e') \in P$ *öyle ki* $e' \neq e$ *and* $d(e') = d(e)$ *biçiminde bir kural yoksa G' deterministiktir.*
İspat: İspat Tanım 7 ve Tanım 8 kullanılarak yapılabilir. □

Teorem 3 (Bir bİ Mutantının Denk Olmaması). *G kullanışlı ise ve* $S \rightarrow e'\ c(e') \in P$ *öyle ki* $e' \neq e$ *ve* $d(e') = d(e)$ *biçiminde bir kural yoksa G', G'ye denk değildir.*
İspat: X ve *Y* Lemma 2'deki gibi tanımlansın:

- $T_{CES}(G) \neq \varnothing$ ve $X \neq \varnothing$ (*G* kullanışlı).
- $Y = \varnothing$ ($S \rightarrow e'\ c(e') \in P$ öyle ki $e' \neq e$ ve $d(e') = d(e)$ biçiminde bir kural yok).

Bu durumda, $d(X) \cap d(Y) = \varnothing$ ve *G', G*'ye denk değildir (Lemma 2). □

Başlangıç işaretle mutantları için seçme taktiği aşağıda verilmektedir.

Başlangıç İşaretle Mutant Seçimi. *Her G' = bİ(G, e) için, aşağıdaki koşulların sağlanması durumunda k-ardışımı e bir mutasyon parametresi olarak seçilmektedir:*
1. x *öyle ki* $d(x_1) = d(e_1)$ *biçiminde bir başlangıç k-ardışımı yoktur.*
1. y *öyle ki* $d(y_1) = d(e_1)$ *biçiminde seçilmiş bir mutasyon parametresi yoktur.*

G'nin kullanışlı ve deterministik bir k-DG olması durumunda yukarıdaki taktik kullanılarak *G*'den üretilen mutantların hepsi kullanışlı ve deterministiktir, ve hiçbiri *G*'ye denk değildir (Teorem 1, Teorem 2 ve Teorem 3). Ayrıca, bu mutantların her biri farklı bir hatayı modellemektedir ve bu hata mutasyon noktasında yer almaktadır: Her *bİ(G, e)* için, e_1 (aynı zamanda $d(e_1)$) bir fazla başlangıç olayıdır.

Seçim dışı bırakılan mutantlar da kullanışlıdır; fakat bu mutantlar ya deterministik değildir ya da daha önce modellenmiş hataları modellemektedir. Bazı deterministik olmayan mutantlar hata modellemeyebilir; hata modellese bile bu hatalardan hiçbiri fazla başlangıç olayı hatası değildir. Bu yüzden bu hataları uç simge ekle mutantları kullanarak modellenmektedir.

Algoritma 1. Başlangıç İşaretleme Mutant Seçimi

Girdi: $G = (E, B; K; C; S; P)$ – k-DG
Çıktı: M – seçilen mutantlar kümesi
$M = \emptyset, N = \emptyset$
for each $b \in B$ **do**
if $S \rightarrow x\ c(x) \in P$ öyle ki $d(x_1) = b$ yoksa **and**
$y \in N$ öyle ki $d(y_1) = b$ yoksa **then**
e = bir k-ardışımı $e \in K$ seç öyle ki $d(e_1) = b$ olsun
$G' = G, M = M \cup \{bİ(G', e)\}, N = N \cup \{e\}$
endif
endfor

Algoritma 1 yukarıdaki taktik kullanılarak başlagıç işaretle mutantlarını seçmektedir. Zaman karmaşası $O(|B|\ |P|)$'dir: (1) Üretilen mutantların sayısı $|B|$ ile sınırlanmaktadır; çünkü her mutantın farklı bir fazla başlangıç olayı hatasını modellemesi istenmektedir. (2) Her *bİ(G, e)*, $O(|P|)$ zamanda mutant seçme taktiğindeki koşulları denetleyerek, ve gerekli atama, kopyalama ve küme birleşimi işlemlerini gerçekleştirerek üretilebilir.

Yukarıdaki taktik kullanılarak üretilen başlangıc işaretle mutantları hatalı 1-ardışımları içermektedir. Seçilen her bir *bİ(G, e)* muntantından *1* uzunluğundaki hatalı tam olay ardışımı (olumsuz sınama örneği) e_1, *O(1)* zamanda üretilebilir.

Örnek 8 (bİ Mutant Seçimi). Şekil 1c'deki 1-DG için seçilen tek başlangıç işaretle mutantı *bİ(G, p1)*'dir. *bİ(G, c1)* ve *bİ(G, x1)* seçilmemektedir; çünkü *c1* ve *x1* başlangıç olaylarıdır. Ayrıca, *bİ(G, p2)* seçilmemektedir; çünkü *bİ(G, p1)* ile aynı hatayı modellemektedir.

3.2 Uç Simge Ekle Mutantları

Uç simge ekle işleci kullanılarak gramere yeni k-ardışımları (uç simgeler) eklenebilir ve eklenen ardışımlar diğerlerine bağlanabilir. Üretilen mutantlar bir olayın bir k-ardışımını izlediği fazla olay hatalarını modellemede kullanılabilir.

Tanım 9 (Uç Simge Ekle (uE)). *Verilen* $e \notin K$ *öyle ki* $d(e) \in B^k$, $U = \{(a, e) | a \in \{a_1, ..., a_m\} \subseteq K\}$ *ve* $V = \{(e, b) | b \in \{b_1, ..., b_n\} \subseteq K \cup \{e\}\}$ *için* $G' = uE(G, e, U, V) = (E, B; K'; C'; S; P')$ *G'nin bir uç simge ekle (uE) mutantıdır;* $K' = K \cup \{e\}$, $C' = C \cup \{c(e)\}$ *ve* $P' = P \cup \{c(e) \rightarrow a\ c(a) | (a, e) \in U\} \cup \{c(b) \rightarrow e\ c(e) | (e, b) \in V\}$.

Amaç küçük sayıda hata modelleyen mutantlar üretmek olduğundan $|U| = 1$ ($U = \{(a, e)\}$) olarak alınmaktadır. Ayrıca, sınamada olumsuz sınama örnekleri olarak sadece hatalı tam olay ardışımları kullanıldığından $V=\varnothing$ olmakta ve $c(e) \rightarrow \varepsilon$ eklenen e k-ardışımının kullanışlılığını sağlamak için kullanılmaktadır.

Örnek 9 (Bir uE Mutantı). Şekil 1c'deki gramere G denilirse, Şekil 3b'deki gramer $uE(G, p3, \{(p2, p3)\}, \varnothing)$ mutantıdır; $p3$, p'ye karşılık gelen yeni bir bağlamlı olaydır. $V = \varnothing$ olduğundan, $c(p3) \rightarrow \varepsilon$, $p3$'ün kullanışlılığını korumakta kullanılmaktadır.

Lemma 3 (Bir uE Mutantının Tam Olay Ardışımları Kümesi). $T_{CES}(G') = T_{CES}(G) \cup \{x\ e |\ S \Rightarrow_G^* x\ c(a)\ (x \in E^*)\}$.
İspat: İspat Tanım 9 kullanılarak ve $c(e) \rightarrow \varepsilon$ kuralı gözetilerek yapılabilir. □

Aşağıda bir uç simge ekle mutantının asıl k-DG'ye denkliği tartışılmaktadır.

Lemma 4 (Bir uE Mutantının Denkliği). *G', G'ye denk değildir ancak ve ancak $d(X) \setminus d(Y) \neq \varnothing$ öyle ki*
- $X = \{x\ e |\ S \Rightarrow_G^* x\ c(a)\ (x \in E^*)\}$ *ve*
- $Y = \{w |\ w \in T_{CES}(G);\ e' \in K,\ w$ *içindedir;* $e' \neq e$ *ve* $d(e') = d(e)\} \subseteq T_{CES}(G)$.

İspat: $T_{CES}(G') = T_{CES}(G) \cup X$ (Lemma 3). Tanım 5 gereği $d(T_{CES}(G')) = d(T_{CES}(G))$ ancak ve ancak $d(X) \subseteq d(Y) \subseteq d(T_{CES}(G))$. □

Bir uç simge ekle mutantının kullanışlılığı, deterministliği ve denk olmaması için yeter koşullar aşağıda verilmektedir.

Teorem 4 (Bir uE Mutantının Kullanışlılığı). *G kullanışlı ise G' kullanışlıdır.*
İspat: İspat Tanım 6, Tanım 9 ve $c(e) \rightarrow \varepsilon$ kuralı gözetilerek yapılabilir. □

Teorem 5 (Bir uE Mutantının Deterministliği). *G deterministik ise ve $c(a) \rightarrow e'\ c(e') \in P$ öyle ki $e' \neq e$ and $d(e') = d(e)$ biçiminde bir kural yoksa G' deterministiktir.*
İspat: İspat Tanım 7, Tanım 9 ve $c(e) \rightarrow \varepsilon$ kuralı gözetilerek yapılabilir. □

Teorem 6 (Bir uE Mutantının Denk Olmaması). *G kullanışlı ve deterministik ise ve $c(a) \rightarrow e'\ c(e') \in P$ öyle ki $e' \neq e$ ve $d(e') = d(e)$ biçiminde bir kural yoksa G', G'ye denk değildir.*
İspat: X and Y Lemma 4'teki gibi tanımlansın:
- $T_{CES}(G) \neq \varnothing$ (G kullanışlı).
- $X \neq \varnothing$ (G kullanışlı, $U \neq \varnothing$ ve $c(e) \rightarrow \varepsilon$ kuralı da eklenmekte).
- $d(X) \cap d(Y) = \varnothing$ ($c(a) \rightarrow e'\ c(e') \in P$ öyle ki $e' \neq e$ ve $d(e') = d(e)$ biçiminde bir kural yoktur ve deterministik G gramerinde her bir boş olmayan başlangıç ardışımına karşılık gelen farklı bir köken olay ardışımı bulunmaktadır.

Bu nedenle, G', G'ye denk değildir (Lemma 4). □

Uç simge ekle mutantları için seçme taktiği aşağıda verilmektedir.

Uç Simge Ekle Mutant Seçimi. *Her $G' = uE(G, e, \{(a, e)\}, \varnothing)$ için, aşağıdaki koşulların sağlanması durumunda $(e, \{(a, e)\}, \varnothing)$ bir mutasyon parametresi olarak seçilmektedir:*
2. *$c(a) \rightarrow x\ c(x) \in P$ öyle ki $d(x_k) = d(e_k)$ biçiminde bir kural yoktur.*
3. *$(y, \{(a, y)\}, \varnothing)$ öyle ki $d(y_k) = d(e_k)$ biçiminde seçilmiş bir mutasyon parametresi yoktur.*

G'nin kullanışlı ve deterministik bir k-DG olması durumunda yukarıdaki taktik kullanılarak *G*'den üretilen mutantların hepsi kullanışlı ve deterministiktir olan ve hiçbiri *G*'ye denk değildir (Teorem 4, Teorem 5 ve Teorem 6). Ayrıca, bu mutantların her biri farklı bir hatayı modellemektedir ve bu hata mutasyon noktasında yer almaktadır: Her *uE(G, e, {(a, e)}, ∅)* için, e_k (aynı zamanda $d(e_k)$) k-ardışımı *a* yı izleyen bir fazla olaydır.

Bu taktik tarafından dışlanan mutantlar kullanışlıdır; fakat bu mutantlar ya deterministik değildir ya da daha önce modellenmiş hataları modellemektedir. Bazı deterministik olmayan mutantlar hata modellemeyebilir; hata modellese bile bu hatalardan hiçbiri mutasyon noktasında yer almamaktadır. Bu yüzden bu hatalar seçilen başka uç simge ekle mutantları tarafından mutasyon noktasında konumlanacak şekilde modellenmektedir.

Algoritma 2. Uç Simge Ekle Mutant Seçimi

Input: $G = (E, B; K; C; S; P)$ – k-DG
Output: M – seçilen mutantlar kümesi
  $M = \varnothing$
  for each $a \in K$ **do**
    $N = \varnothing$
    for each $b \in B$ **do**
      if $c(a) \rightarrow x\ c(x) \in P$ öyle ki $d(x_k) = b$ yoksa **and**
        $(y, (a, y), \varnothing) \in N$ öyle ki $d(y_k) = b$ yoksa **then**
          $b' = b$'nin yeni bir bağımlı olayı, $e = a_2 \ldots a_k\ b'$
          $G' = G$, $M = M \cup \{uE(G', e, \{(a, e)\}, \varnothing)\}$, $N = N \cup \{e\}$
      endif
    endfor
  endfor

Algoritma 2 yukarıdaki taktik kullanılarak uç simge ekle mutantlarını seçmektedir. Zaman karmaşası $O(|K|\ |B|\ |P|)$'dir: (1) Üretilen mutantların sayısı $|K|\ |B|$ ile sınırlanmaktadır; çünkü her mutantın farklı bir fazla olay hatasını modellemesi istenmektedir. (2) Her *uE(G, e, {(a, e)}, ∅)*, $O(|P|+|B|+k) = O(|P|)$ zamanda mutant seçme taktiğindeki koşulları denetleyerek, ve gerekli atama, kopyalama ve küme birleşimi işlemlerini gerçekleştirerek üretilebilir.

Yukarıdaki taktik kullanılarak üretilen uç simge ekle mutantları hatalı (k+1)-ardışımları içermektedir. Seçilen her bir *uE(G, e, {(a, e}, ∅)* mutantından $a\ e_1$ hatalı (k+1)-ardışımını içeren bir hatalı tam olay ardışımı (olumsuz sınama örneği) derinlik öncelikli arama kullanılarak $O(|P|)$ zamanda üretilebilir.

Örnek 10 (uE Mutant Seçimi). Şekil 1c'deki gramer *G* olsun. Mutant seçimi için kullanılabilecek tek köken olayı *p*'dir; çünkü *c* ve *x* tüm olayları takip edebilmektedir. Bu durumda seçilen tek mutant *uE(G, p3,{(p2, p3}, ∅)* olmaktadır; çünkü *p2*'yi izleyen bir *p* olayı bulunmamaktadır.

4 Örnek Çalışma

Bu kısımda, öne sürülen yaklaşımın gerçekçi bir değerlendirmesini yapmak için olay

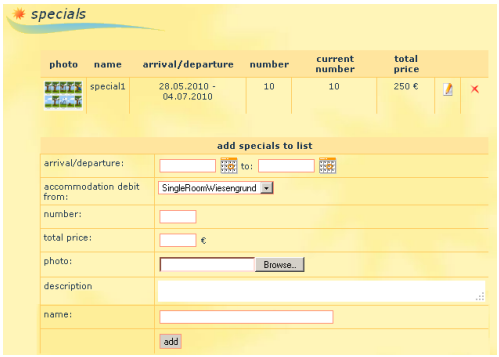
tabanlı sınama için daha önce öne sürülmüş olan ve *olay ardışım çizgelerini* (*OAÇ*) kullanan model tabanlı mutasyonla sınama yaklaşımıyla [10][14] gerçek bir sistem üzerinde karşılaştırmalı uygulaması gerçekleştirilmektedir.

4.1 Yaklaşımlar ve Parametreleri

Bu bildiride öne sürülen yaklaşım *k-DG* olarak adlandırılmakta ve OAÇ modeli kullanımına dayanan yaklaşıma *OAÇ(k, k)* denmektedir. *k=1,2,3* olarak seçilmekte ve her bir k-DG, *OAÇ(k+1, k)* ile eşleştirilmektedir.

k-DG yaklaşımında sistemin k-DG modeli olumlu sınama örnekleri üretmede [6][7], ve Kısım 3.1'deki ve Kısım 3.2'deki taktiklerle üretilen mutantlar olumsuz sınama örnekleri üretmede kullanılmaktadır.

OAÇ(k+1, k) yaklaşımı verilen OAÇ modelinden elde edilen mutantlardan k-ardışım kapsamasını sağlayacak şekilde sınama örnekleri üretmeye dayanmaktadır. Bir sistemin OAÇ modeli o sistemin 1-DG modelinin köken olayları çıkartılmış hali olarak düşünülebilir (Örnek: Şekil 1b). Yaklaşımda Kısım 3'ün başında adı geçen 8 adet mutasyon işleci kullanılmaktadır (Ayrıntılar için: [10][14]). Tüm mutantlar kullanıldığında toplamda çok fazla sayıda mutant ve dolayısıyla sınama örneği üretilmektedir (Bakınız: **Hata! Başvuru kaynağı bulunamadı.**). Bu yüzden, bu örnek çalışmada her bir *OAÇ(k+1, k)* için: Önce, karşılık gelen *k-DG* yaklaşımıyla üretilen sınama kümesinin toplam büyüklüğü *b* olarak seçilmekte; daha sonra, (k+1)-ardışımları kapsayarak mutantlardan üretilen sınama örneklerinin sistem üzerinde işletilebilecek toplam büyüklüğü *b* değerini aşana kadar rastgele OAÇ mutantları seçilmektedir. Bununla *k-DG* ve *OAÇ(k+1, k)* ile üretilen sınama örneklerinin işletilmesinde harcanan çabaların birbirine yakın tutulması amaçlanmaktadır.



Şekil 4. `Specials` ana sayfası (kullanıcı arayüzü).

4.2 Sınama Altındaki Sistem, Sistem Modelleri ve Mutant Sayıları

ISELTA otel sahipleri ve seyahat acenteleri için tasarlanmış turistik amaçlı ve ticari bir çevrimiçi yer ayırtma sistemidir. Bu çalışmada ISELTA'nın `Specials` modülü

seçilmiştir. (Arayüz: Şekil 4; modülle ilgili ayrıntılar için: [7][8])

Hata! Başvuru kaynağı bulunamadı. örnek çalışmada kullanılan modellerin büyüklüklerini ve üretilen mutantların toplam sayılarını göstermektedir.

Table 1. Model büyüklükleri ve toplam mutant sayıları.

	1-DG	2-DG	3-DG	OAÇ
Kurallar	429	2191	11205	-
k-ardışımları	90	427	2184	-
Kenarlar	-	-	-	429
Olaylar	-	-	-	90
Toplam Mutantlar	755	3378	17732	737370

Hata! Başvuru kaynağı bulunamadı. aynı zamanda Kısım 4.1'de tanımlanan *OAÇ(k+1, k)* yaklaşımına *k* parametresi eklenerek üretilen sınama kümesinin boyutlarının sınırlandırılmasının sebebini de göstermektedir: *k-DG* yaklaşımlarının mutant sayıları toplansa bile OAÇ modelinden üretilen mutant sayısının ~%2,97'sine ulaşılmaktadır. Bu aynı zamanda Kısım 3'te öne sürülen mutant seçme taktiklerinin mutant sayıları ne kadar önemli oranda azalttığını göstermektedir.

4.3 Hata Ekimi

Sınama yaklaşımlarını gerçekçi olarak karşılaştırmak için rastgele seçilen hatalar ekmek ola gelen bir tekniktir [18][13]. Bu çalışmada ekilen hataları modellemek için farklı m-DG modelleri *m=1,2,3,4* kullanılmaktadır. Bunun nedeni tespit etme zorluğu farklı seviyelerde olan hatalar üretmektir. Genel olarak, *m* değeri artıkça modellenen hataların tespit edilmesi zorlaşmaktadır; çünkü kapsanması gereken ardışımların uzunlukları ve sayıları artmaktadır. Her *m* değeri için 25'i eksik olay hatası ve 25'i fazla olay hatası olmak üzere 50 hata ekilmiştir. Bu tüm *m* değerleri için toplamda 200 hata etmektedir.

Bu doğrultuda yaklaşımları değerlendiriken gözden kaçırılmaması gerek bir nokta şudur: Sabit bir *k* değeri için *k-DG* ve *OAÇ(k+1, k)* yaklaşımları temelde $m \leq k$ olan m-DG'ler tarafından modellenen hataları bulmayı hedeflemektedir.

4.4 Sınama Sonuçları

Sınama kümelerinin işletimi izleyen şekilde gerçekleştirilmektedir: Her sınama örneği bir aksaklık gözlenene veya örnek sonuna ulaşılana dek işletilir. Bir aksaklık gözlendiğinde bu aksaklığa karşılık gelen hata bulunup düzeltilir ve işletim bu hatayı tespit eden örneğin yeninden işletilmesiyle devam eder. Herhangi bir örnek hiç bir aksaklıkla karşılaşılmadan sonuna kadar işletildikten sonra bir daha hiç işletilmez. Bu süreç sınama kümesindeki tüm örnekler sonuna dek işletilene kadar devam eder.

Tablo 1 her bir yaklaşım için sınama kümesi üretme ve işletme sonuçlarını özetlemektedir. Ayrıca, sınama kümelerinin işletim sürecinde bulunan hata sayılarının işletilen olay sayılarına göre nasıl değiştiğini gösteren sınama eğrileri Şekil 5'te verilmektedir.

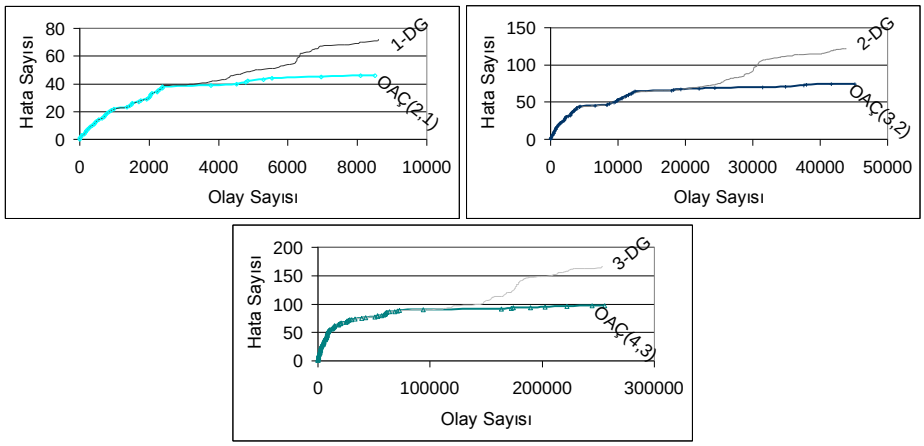
Tablo 1. Sınama kümesi üretme ve işletme sonuçları.

Sınama Kümesi	Sınama Örneği	İşletilebilinir Uzunluk	Üretme Zamanı (s)	Toplam İşletilen	Bulunan Hata
1-DG	832	7387	1	8613	72
OAÇ(2, 1)	372	7419	10	8501	46
2-DG	3972	41548	4	43851	122
OAÇ(3, 2)	2187	42910	44	45146	74
3-DG	21438	250383	58	253668	166
OAÇ(4, 3)	15313	253109	1321	255789	97

4.5 Sonuçların Yorumu

Tablo 1'deki veriler kullanılarak aşağıdaki yorumlar yapılabilir.

- Hata Bulma Etkinliği: *k-DG* yaklaşımları, *OAÇ(k+1, k)* yaklaşımlarına göre %56,52 - %71,13 daha fazla hata bulmuştur; *k-DG* hata bulmada daha etkindir.
- Sınama Üretme Maliyeti: *OAÇ(k+1, k)* sınama kümelerinin üretilmesi için gereken zaman *k-DG* sınama kümelerinin üretilmesi için geçen zamanın 10 - 23 katıdır. Bu *k-DG*'nin sınama kümesi üretmede daha etkin olduğunu gösterir.
- Hata Bulma Verimi: Hata bulma etkinliklerine paralel olarak, *k-DG* yaklaşımlarının sınama sonundaki hata bulma oranları (bulunan_hata_sayısı / işletilen_olay_sayısı) *OAÇ(k+1, k)* yaklaşımlarınını hata bulma oralnlarının 1.54 - 1.73 katıdır. Yani, *k-DG* yaklaşımları hata bulmada daha verimlidir.



Şekil 5. Sınama eğrileri.

Ayrıca, Şekil 5'teki sınama eğrileri bize şunları göstermektedir:

- Sınama işletim eğilimi, *k* değerine bağlı olarak, işletim sürecinin ilk %28,34'lük (*k=1*), %46,16'lık (*k=2*) ve %63,99'luk (*k=3*) kısmı boyunca *k-DG* ve *OAÇ(k+1, k)* yaklaşımları için hemen hemen tamamıyla aynıdır.

- Bu kısım aynı zamanda *k-DG* yaklaşımlarıyla sınamayla bulunan toplam hataların %55,56 - %56,56'sının ortaya çıkarıldığı zamana denk gelmektedir.
- Bu noktadan sonra *k-DG* yaklaşımları *OAÇ(k+1, k)* yaklaşımlarına göre hata bulma yönünden gittikçe daha etkin ve verimli olmaktadır.

4.6 Geçerliliği Tehdit Edebilecek Noktalar

Örnek çalışmanın doğası gereği elde eliden sonuçların geçerliliğini tehdit edebilecek bazı noktalar bulunmaktadır.

Sonuçlar sadece bir sistem üzerinde elde edilmiştir. Sistemin kendine özgü karakterinin sonuçlar üzerinde etkisi olabileceğinden değişik tipte ve birden fazla sistem üzerinde benzer örnek çalışmalar yapılması güvenirliği artıracaktır.

OAÇ tabanlı yaklaşım üretilen sınama kümesinin büyüklüğünü sınırlandıracak şekilde değişime uğratılmıştır. Bu yaklaşımın tam anlamıyla uygulanmamış olması demektir. Fakat böyle bir sınır kullanılmadığı takdirde sistemdeki tüm hatalar bulunsa bile çok fazla sayıda mutant ve sınama örneği üretileceğinden sınama sürecinin verimi aşırı derecede düşecektir. Bunu yapmak yerine daha yüksek *k* değerlerine sahip *k-DG* yaklaşlaşımlarını kullanmak çok daha akılcı olacaktır.

5 Sonuç

Bu bildiride model tabanlı mutasyonla sınama için geliştirlen yeni bir olay tabanlı sınama yaklaşımı ele alınmaktadır. Yaklaşımın var olan OAÇ tabanlı yaklaşımdan en bazı büyük farkları içerdiği mutant seçme taktikleri ve buna bağlı olarak kullanılan sınama kümesi üretme şeklidir.

Yapılan örnek çalışmanın gösterdiği üzere öne sürülen k-DG tabanlı yaklaşım ve var olan OAÇ tabanlı yaklaşım sınama işletim maliyetleri (sınama kümelerinin işletilebilir büyüklükleri) dengelenecek şekilde uygulandıklarında k-DG tabanlı yaklaşımla gelen iyileştirmeler belirgin olarak görülmektedir:

- k-DG tabanlı yaklaşım OAÇ tabanlı yaklaşımdan %73'e kadar daha verimli olabilmektedir.
- k-DG tabanlı yaklaşımla sınama kümesi üretimi OAÇ tabanlı yaklaşımla sınama kümesi üretiminden 22 kata kadar daha hızlı olabilmektedir.

Gelecek çalışmalarda, yaklaşımın üretilen sınama kümelerinin işletilebilir büyüklükleri yerine kapsama kriterlerini dengeleyici etmen olarak kullanarak rastgele sınamayla karşılaştırılması düşünülmekte ve sonuçların güvenilirliğini artırmak için ek sistemlerin de çalışmaya dahil edilmesi planlanmaktadır.

Kaynaklar

1. Adamopoulos, K., Harman, M., Hierons, R.: How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. In: AAAI Genetic

and Evolutionary Computation Conference 2004 (GECCO 2004), Lecture Notes in Computer Science, vol. 3103, pp. 1338-1349, Springer, Heidelberg (2004)
2. Aichernig, B.K.: Mutation Testing in the Refinement Calculus. Formal Aspects of Computing Journal, 15(2-3), 280-295 (2003)
3. Ammann, P.E., Black, P.E., Majurski, W.: Using Model Checking to Generate Tests from Specifications. In: Proceedings of the 2nd International Conference on Formal Engineering Methods (ICFEM 1998), pp. 46-54, IEEE Computer Society, Washington DC (1998)
4. Beizer, B.: Software Testing Techniques. Van Nostrand Reinhold, New York (1990)
5. Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. In: Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE 2001), pp. 34-43, IEEE Computer Society, Washington DC (2001)
6. Belli, F., Beyazıt, M.: A Formal Framework for Mutation Testing. In: Proceedings of the 2010 4th International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2010), pp. 121-130, IEEE Computer Society, Washington DC (2010)
7. Belli, F., Beyazıt, M.: Grammar Based Mutation Testing. In: Proceedings of the 5th National Software Engineering Symposium (V. Ulusal Yazılım Mühendisliği Sempozyumu - UYMS 2011), pp. 38-45, Ankara (2011) (in Turkish)
8. Belli, F., Beyazıt, M., Güler, N.: Event-Oriented, Model-Based GUI Testing and Reliability Assessment—Approach and Case Study. Advances in Computers, 85, 277-326 (2012)
9. Belli, F., Beyazıt, M., Memon, A.: Testing is an Event-Centric Activity. In: Proceedings of the 6th International Conference on Software Security and Reliability (SERE-C 2012), pp. 198-206, IEEE Computer Society, Washington DC (2012)
10. Belli, F., Budnik, C.J., Wong, W.E.: Basic operations for generating behavioral mutants. In: Proceedings of the 2nd Workshop on Mutation Analysis (MUTATION 2006), pp. 9-18, IEEE Computer Society, Washington DC (2006)
11. Budd, T.A., Gopal, A.S.: Program Testing by Specification Mutation. Computer Languages, 10(1), 63-73 (1985)
12. Grün, B.J.M, Schuler, D., Zeller, A.: The Impact of Equivalent Mutants. In: International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2009), pp. 192-199, IEEE Computer Society, Washington DC (2009)
13. Harrold, M.J., Offutt, A.J., Tewary, K: An approach to fault modeling and fault seeding using the program dependence graph. Journal of Systems and Software 36(3), 273-295 (1997)
14. Hollmann, A.: Model-Based Mutation Testing for Test Generation and Adequacy Analysis. Ph.D. Thesis, University of Paderborn, Paderborn (2011)
15. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, 3rd Edition. Addison-Wesley, Boston MA (2006)
16. Klint, P., Lämmel, R., Verhoef, C.: Toward an engineering discipline for grammarware. ACM Transactions on Software Engineering and Methodology, 14(3), 331-380 (2005)
17. Offutt, A.J., Ammann, P., Liu, L.: Mutation Testing Implements Grammar-Based Testing. In: Proceedings of the 2nd Workshop on Mutation Analysis (MUTATION 2006), pp. 12-21, IEEE Computer Society, Washington DC (2006)
18. Offutt, A.J., Hayes, J.H.: A semantic model of program faults. In: Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 1996), pp. 195-200, ACM, New York NY (1996)
19. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco (2006)
20. Zhu, H., Hall, P.A., May, J.H.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4), 366-427 (1997)