Hybrid Concolic Testing

در اجرای دلخواه می‌توان به حالت‌هایی با عمق زیاد از برنامه دست یافت ولی نمی‌توان به صورت گسترده و با پوشش بالا کد را اجرا کرد.

اجرای نمادین و Concolic می‌توانند به پوشش بالای کد دست یابند ولی در تعداد بالای اجرا نمی‌توانند به حالت‌های با عمق بالا برسند.

اجرای هیبرید به صورت ترکیبی اجرای دلخواه و Concolic را انجام می‌دهد تا بتواند از مزیت‌های هر یک استفاده کند. کار ارائه شده بروی ابزار CUTE هست و تمام مفاهیم و اجراهای ذکر شده یعنی اجرای عددی یا Concolic مطابق با کار ارائه شده CUTE در سال ۲۰۰۵ هست.

ابتدا کد به صورت عددی اجرا می‌شود. هر گاه اجرا اشباع شد اجرا به Concolic تغییر میابد تا بتواند به صورت عمق‌محدود به پوشش بیشتری از کد برسد. دوباره بعد از یافتن مسیر جدید اجرا به عددی تغییر میابد.

برای مثال در text editorها ابتدا به صورت دلخواه تعدادی ورودی (متن) تولید می‌شود تا بتوان عملگرهای ,delete insert و ... را انجام داد.

اجرای هیبرید برای برنامه‌های تعاملی مثل برنامه‌های رخدادمحور یا دارای GUI مناسب است.

این اجرا همان محدودیت‌های اجرای Concolic را دارد. ممکن است به پوشش ۱۰۰ درصد از کد نرسد ولی از نظر نویسندگان پوشش کامل نشانه ای برای قابل اعتماد بودن[۱] کد نیست.

برای مثال در کد شکل یک اگر به تنهایی به صورت دلخواه اجرا کنیم با احتمال 10^{-7} می‌توان به خطا رسید.

اگر به صورت Concolic تنها اجرا شود، 10^{11} اجرا نیاز است تا به خطا برسد.

ولی در اجرای هیبرید بعد از حدود ۲ دقیقه که برنامه به شکل دلخواه اجرا شد و به state=9 رسید با یک اجرا می‌توان رشته reset را تولید کرد و به خطا رسید.

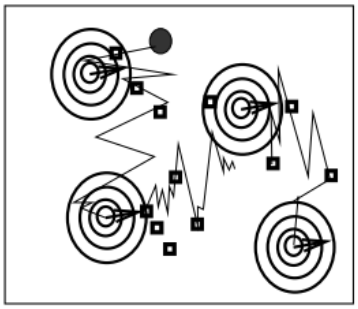
شکل ۳ نمای غیر فورمال از اجرای هیبرید را نشان می‌دهد در این جا ابتدا به صورت دلخواه اجرا می‌شود. (خطوط شکسته)

```
void testme() {
  char * s;
  char c;
  int state = 0;

  while (1) {
      c = input();
      s = input();

     /* a simple state machine */
      if (c == '[' && state == 0) state = 1;
      if (c == '(' && state == 1) state = 2;
      if (c == '{' && state == 2) state = 3;
      if (c == '~' && state == 3) state = 4;
      if (c == 'a' && state == 4) state = 5;
      if (c == 'x' && state == 5) state = 6;
      if (c == '}' && state == 6) state = 7;
      if (c == ')' && state == 7) state = 8;
      if (c == ']' && state == 8) state = 9;

      if (s[0] == 'r' && s[1] == 'e'
          && s[2] == 's' && s[3] == 'e'
          && s[4] == 't' && s[5] == 0
          && state == 9) {
        ERROR;
```

شکل ۱: نمونه کد

[۱] Reliability

بعد از این که هدف (مربع‌های کوچک) جدیدی یافت نشد به صورت محلی Concolic اجرا می‌شود .(دایره‌های تودرتو)



شکل ۳: نمای غیر فورمال از اجرای هیبرید

در شکل ۲ در خطوط ۴ تا ۱۶ یک حلقه وجود دارد که بررسی می‌کند که کد به اندازه θ_2 به صورت عددی و دلخواه اجرا شود.

هرگاه تعداد اجراها به θ_2 رسید دستور fork اجرا می‌شود و حالت فعلی ذخیره می‌شود. سپس پردازه فرزند به صورت Concolic اجرا می‌شود. بعد از اتمام اجرای Concolic پردازه فرزند kill شده و پردازه پدر که در حال انتظار بوده است به اجرای خود ادامه می‌دهد و حالت ذخیره شده بازیابی می‌شود و دوباره به خط ۳ پرش می‌کند.

Algorithm 2 Algorithm HCT

Input: program P, set of coverage goals Goals.

```
1:  while Goals ≠ ∅ do
2:     ℓ = ℓ0, M = M0, IMap = Random
3:     iter = 0
4:     while iter < θ2 or stmt_at(ℓ) is not x := input() do
5:        if stmt_at(ℓ) = halt then
6:           break
7:        if stmt_at(ℓ) = bug then
8:           return bug
9:        (ℓ, M) = Concrete(ℓ, M, IMap)
10:       remove covered goals from Goals
11:       if coverage has increased then
12:          iter = 0
13:       else
14:          if stmt_at(ℓ) is x := input() for some x then
15:             iter = iter + 1
16:    endwhile
17:    if iter = θ2 then
18:       snapshot(M)
19:       IMap = Concolic(ℓ, M)
20:       M = restore()
21:       goto 3
22: endwhile
```

شکل ۲: کد مربوط به اجرای هیبرید