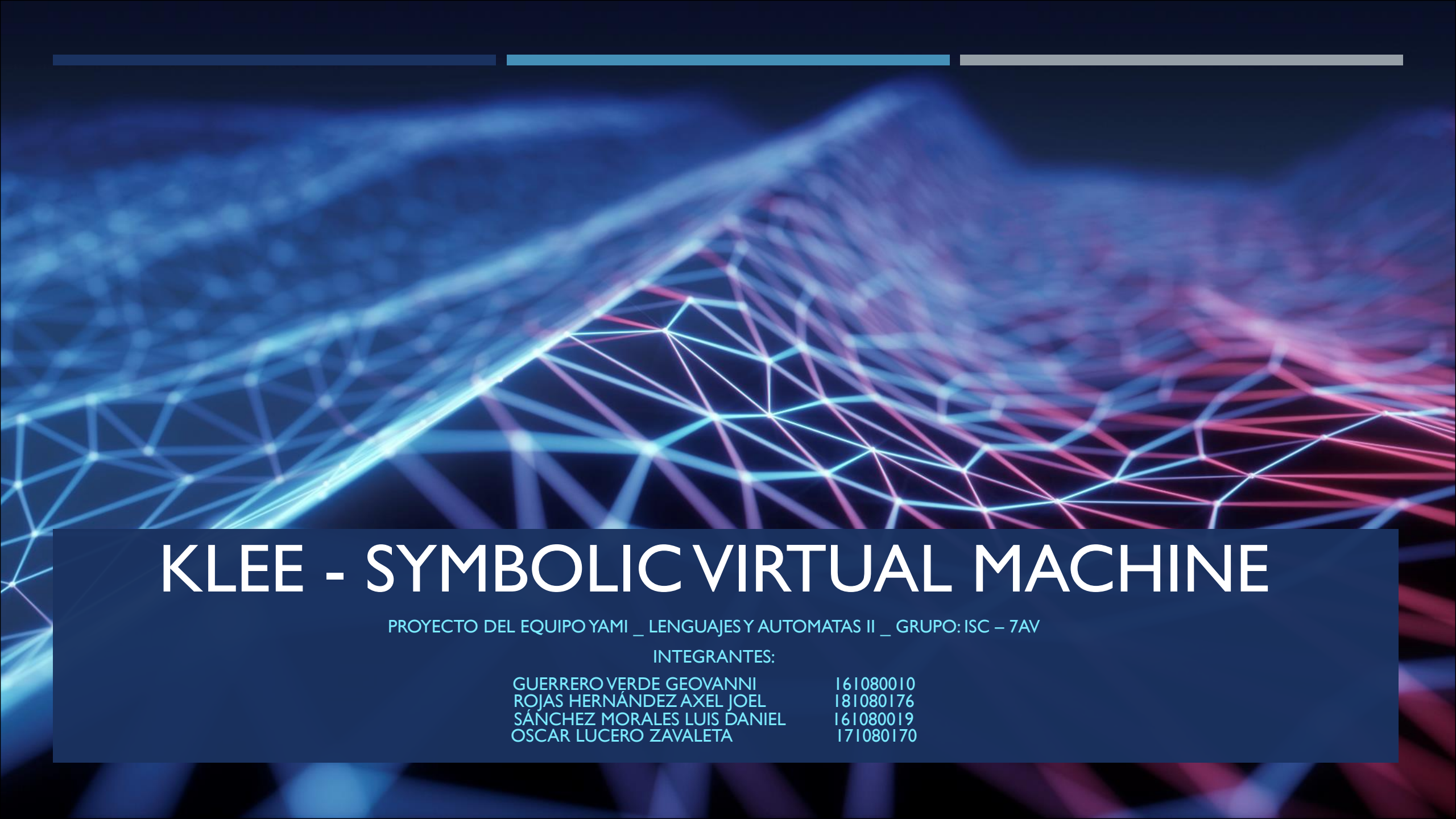


KLEE - SYMBOLIC VIRTUAL MACHINE

PROYECTO DEL EQUIPO YAMI _ LENGUAJES Y AUTOMATAS II _ GRUPO: ISC – 7AV

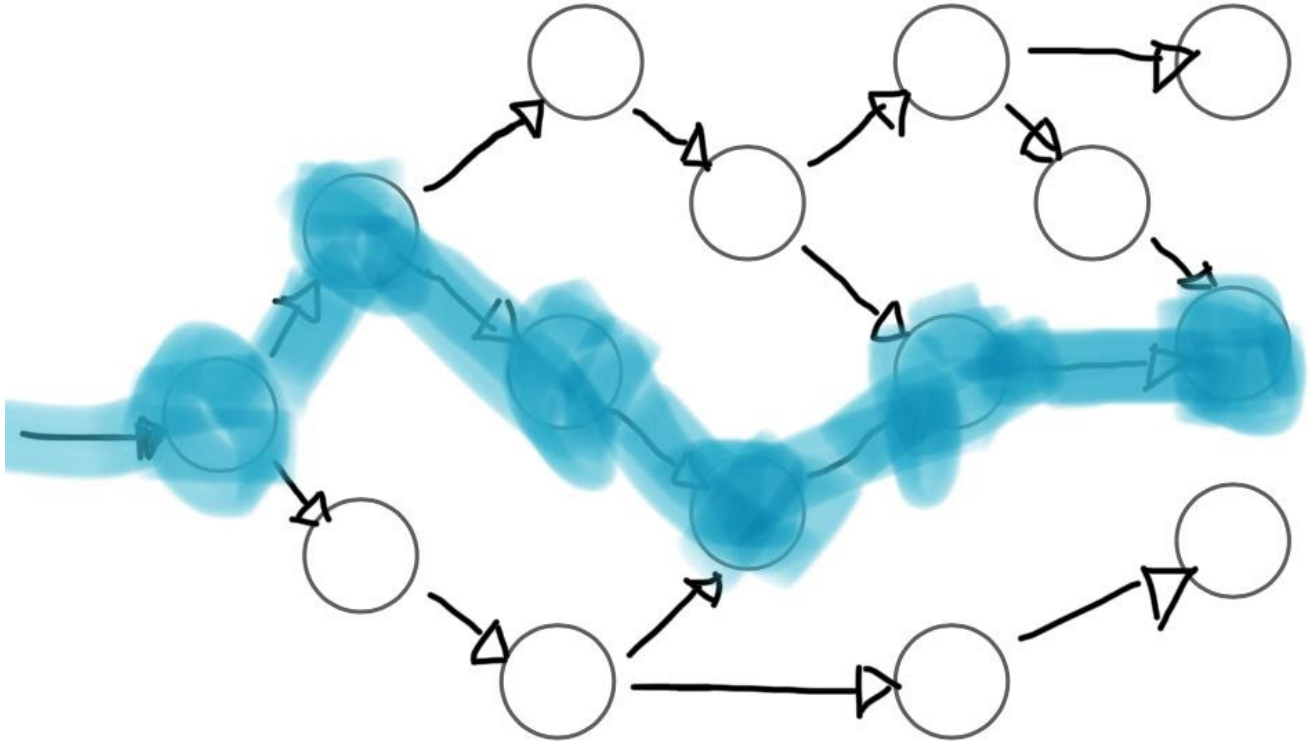
INTEGRANTES:

GUERRERO VERDE GEOVANNI	161080010
ROJAS HERNÁNDEZ AXEL JOEL	181080176
SÁNCHEZ MORALES LUIS DANIEL	161080019
OSCAR LUCERO ZAVALETA	171080170

¿QUÉ ES KLEE?

Klee es un motor de ejecución simbólica dinámica.

¿Que es la ejecución simbólica?
Le llamamos ejecución simbólica a probar el software sin tener datos reales, una prueba con variables.
Por ejemplo, si hiciéramos un software para un banco, a la hora de hacer pruebas no podemos testear con la información monetaria de los clientes de la empresa…
Tendríamos que usar en este caso un software para probar que todas las transacciones o acciones realizadas cumplieran con la seguridad requerida.



Si yo quiero recorrer un camino en especifico, necesito poder descubrir qué características deben tener los datos de entrada para forzar dicho camino.

¿CÓMO FORZAMOS UN CAMINO?

PC (Path condition)

El PC representa la condición booleana de ejecución en su camino, Su valor, es una expresión simbólica booleana.

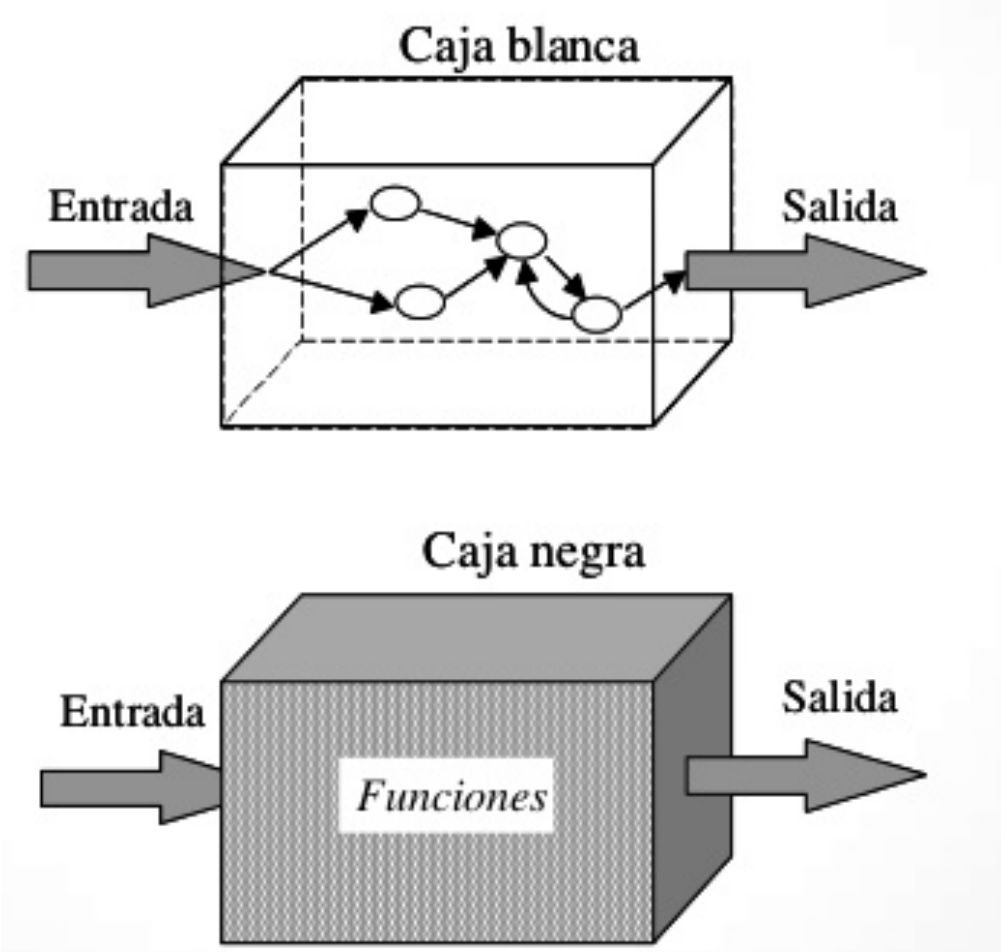
La condición de ejecución de un camino permite saber si el camino es ejecutable. Permite derivar los datos que causan la ejecución de dicho camino.

1. PROGRAM ATP	1. { (x = undef), (x = undef), (PC = true) }
2. VAR	2.
3. x, p : INTEGER;	3.
4. BEGIN	4.
5. read(x);	5. { (x = Q), (p = undef), (PC = true) }
6. read(p);	6. { (x = Q), (p = A), (PC = true) }
7. p = ABS(p);	7. { (x = Q), (p = |A|), (PC = true) }
8. x = p;	8. { (x = |A|), (p = |A|), (PC = true) }
9. IF (x >= 0)	9. [(x >= 0)] -> [(|A| >= 0)] -> Siempre true
10. THEN write(x);	
11. ELSE write(0);	
12. END	

USADO EN HACKING

Las pruebas de caja blanca y caja negra son metodologías que estudian las entradas y salidas de datos.

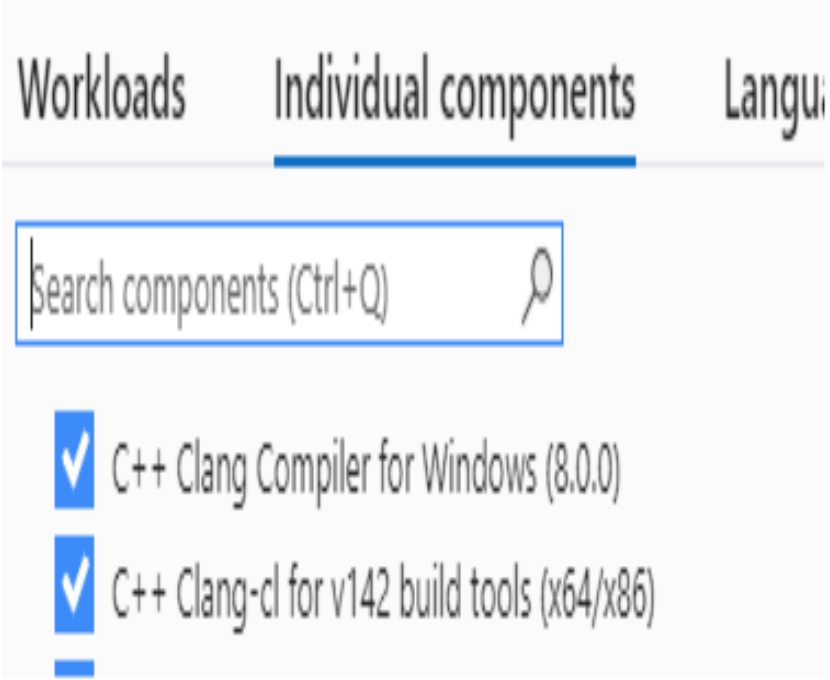
Las pruebas de caja blanca se centran en entender la salida de los datos teniendo en cuenta su funcionamiento interno, mientras que las de caja negra solo se centran en los resultados, sin importarles el funcionamiento interno.



PRUEBAS DEL PRIMER TUTORIAL

Instalar en pc

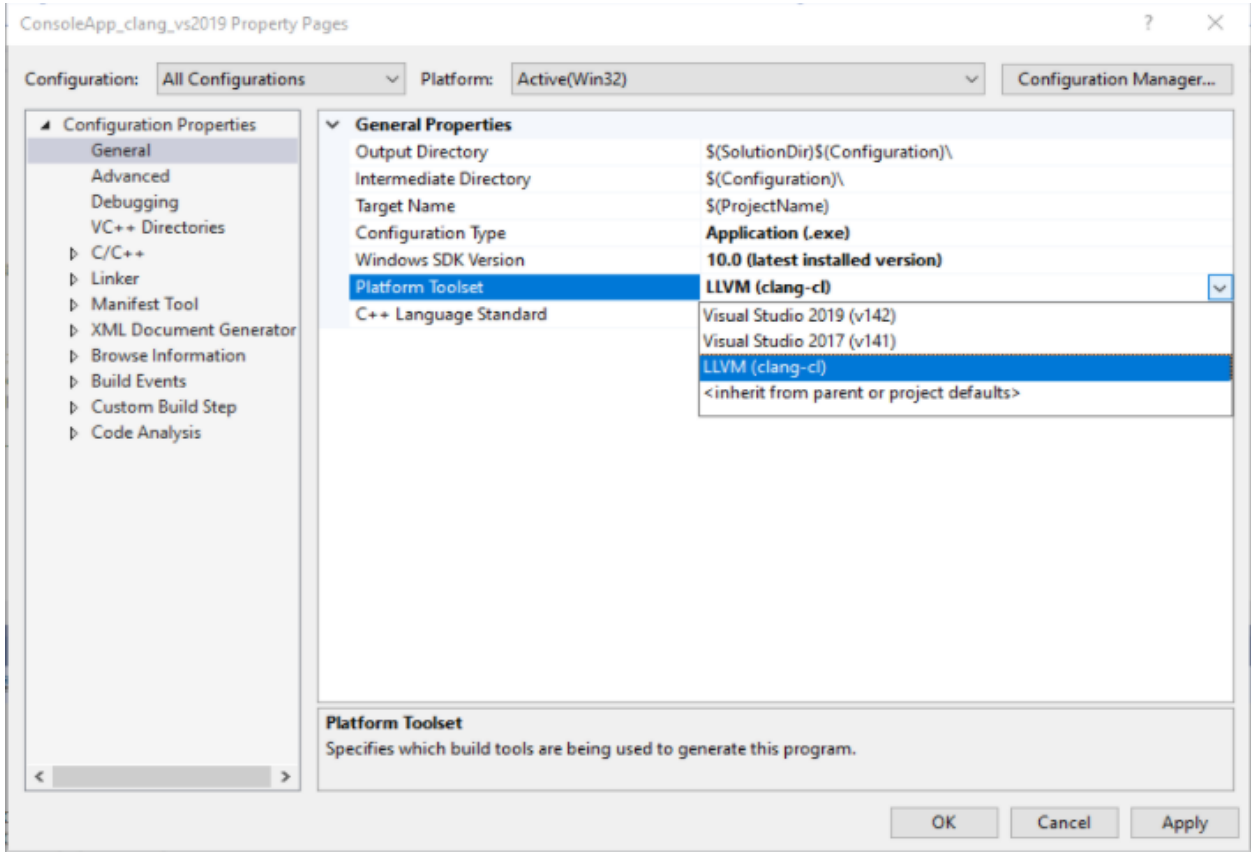
Para obtener la mejor compatibilidad con IDE en Visual Studio, recomendamos utilizar las últimas herramientas de compilación de Clang para Windows. Si aún no los tiene, puede instalarlos abriendo el instalador de Visual Studio y eligiendo las herramientas C ++ Clang para Windows en Desarrollo de escritorio con componentes opcionales de C ++ . Si prefiere utilizar una instalación de Clang existente en su máquina, elija C ++ Clang-cl para las herramientas de compilación v142. componente opcional. La biblioteca estándar de Microsoft C ++ actualmente requiere al menos Clang 8.0.0; la versión empaquetada de Clang se actualizará automáticamente para mantenerse al día con las actualizaciones en la implementación de Microsoft de la Biblioteca estándar.



PRUEBAS DEL PRIMER TUTORIAL

Configurar un proyecto de Windows para usar herramientas Clang

Para configurar un proyecto de Visual Studio para usar Clang, haga clic con el botón derecho en el nodo del proyecto en el Explorador de soluciones y elija Propiedades . Por lo general, primero debe elegir Todas las configuraciones en la parte superior del cuadro de diálogo. Luego, en General > Conjunto de herramientas de plataforma , elija LLVM (clang-cl) y luego Aceptar .



PRUEBAS DEL PRIMER TUTORIAL

Si está utilizando una instalación personalizada de Clang, puede modificar Proyecto > Propiedades > DIrectorios VC ++ > Propiedades de configuración > Directorios ejecutables agregando la raíz de instalación personalizada de Clang como el primer directorio allí, o cambiar el valor de la LLVMInstallDirpropiedad. Consulte Establecer una ubicación LLVM personalizada para obtener más información.

Configurar un proyecto de Linux para usar herramientas Clang

Para proyectos de Linux, Visual Studio usa la interfaz compatible con Clang GCC. Las propiedades del proyecto y casi todos los indicadores del compilador son idénticos

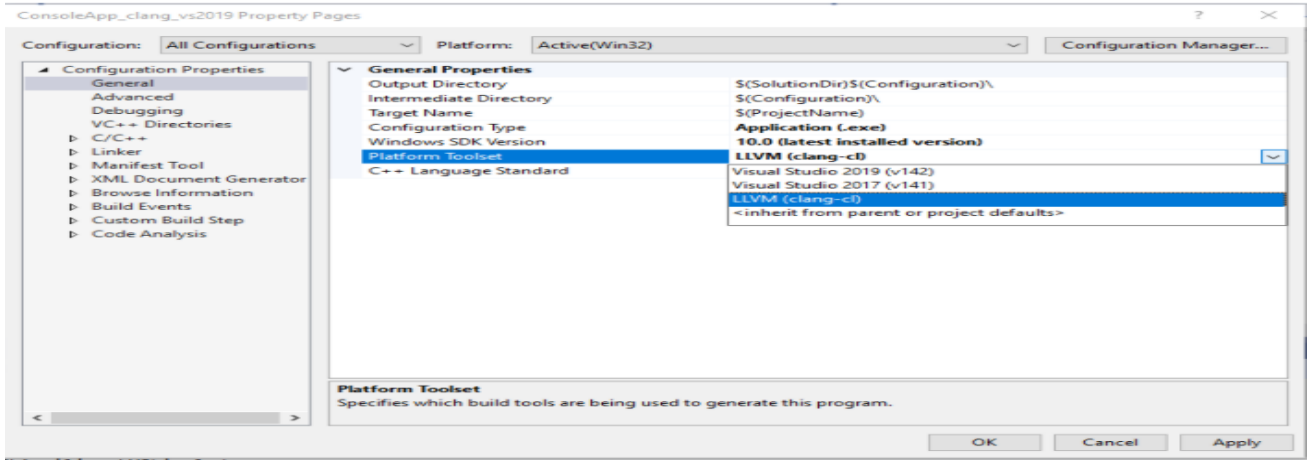
Para configurar un proyecto de Visual Studio Linux para usar Clang:

Haga clic con el botón derecho en el nodo del proyecto en el Explorador de soluciones y elija Propiedades.

Por lo general, primero debe elegir Todas las configuraciones en la parte superior del cuadro de diálogo.

En General > Conjunto de herramientas de plataforma , elija WSL_Clang_1_0 si está usando el Subsistema de Windows para Linux, o Remote_Clang_1_0 si está usando una máquina o VM remota.

COMO MARCAR LA PARTE DEL CÓDIGO QUE NOS INTERESA…



Establecer una ubicación LLVM personalizada

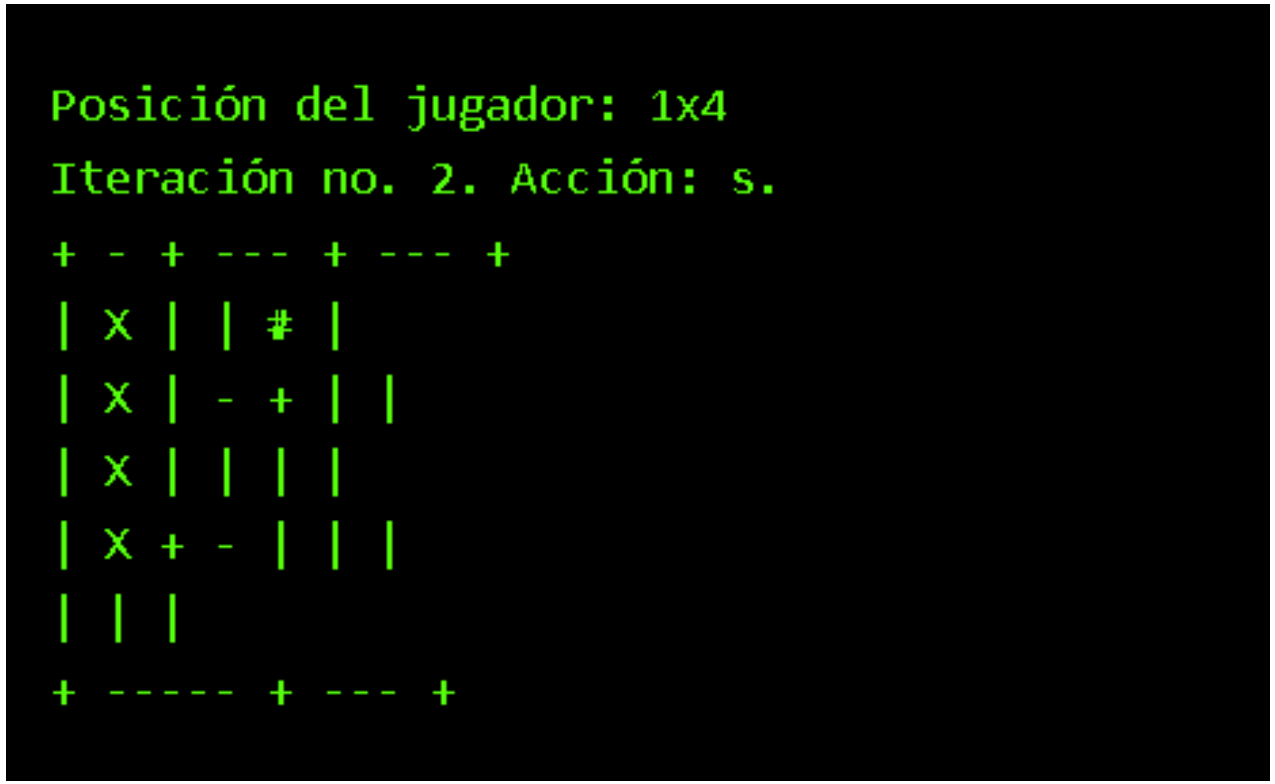
Puede establecer una ruta personalizada para LLVM para uno o más proyectos creando un archivo *Directory.build.props* y agregando ese archivo a la carpeta raíz de cualquier proyecto. Puede agregarlo a la carpeta raíz de la solución para aplicarlo a todos los proyectos de la solución. El archivo debería verse así (pero sustituya su ruta real):

XML Copiar

```xml
<Project>
  <PropertyGroup>
    <LLVMInstallDir>c:\MyLLVMRootDir</LLVMInstallDir>
  </PropertyGroup>
</Project>
```

PRUEBAS DEL SEGUNDO TUTORIAL

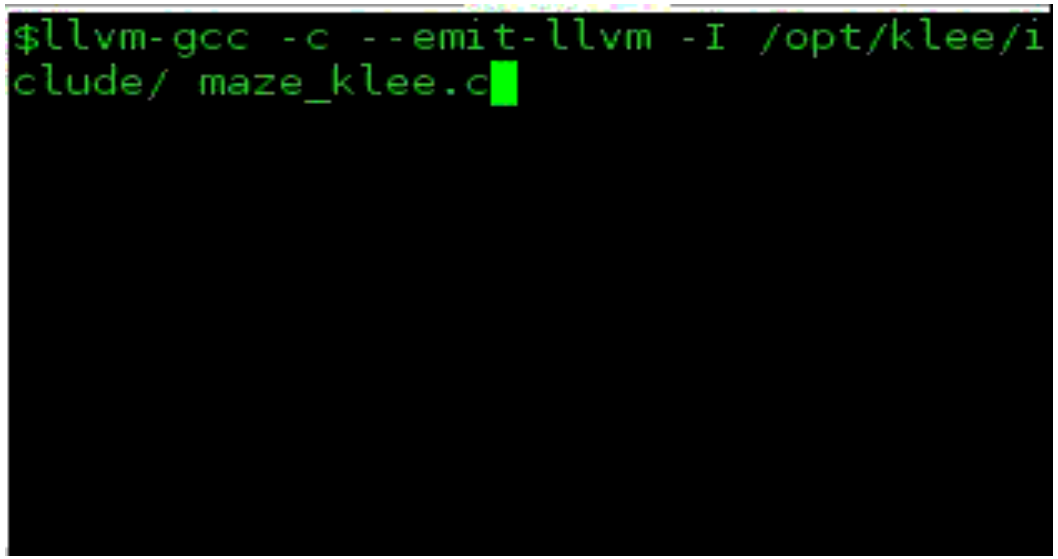
Resolviendo un laberinto



PRUEBAS DEL SEGUNDO TUTORIAL

El camino mas obvio: Este simple juego ASCII te pide que primero lo alimentes con instrucciones. Debe ingresarlos como una lista de acciones por lotes. Como siempre"; a es izquierda, d es derecha, w es arriba y s es abajo

Pruebas con klee: Ahora KLEE encontrará todos los códigos / rutas de laberinto posibles accesibles desde cualquier entrada. Si algunas de esas rutas conducen a una condición de error típica, como una falla de memoria o algo así, ¡KLEE lo indicará!



PRUEBAS DEL SEGUNDO TUTORIAL – MARCANDO LA PARTE DEL CÓDIGO QUE NOS INTERESA

En el código, eso se hace reemplazando esta línea...

```
printf ("¡Tú ganas! \ n");
```

Por estas dos:

```
printf ("¡Tú ganas! \ n");
klee_assert (0); // Señal ¡¡La solución !!
```

```
$ ls -1 klee-last / | grep -A2 -B2 err
test000096.ktest
test000097.ktest
test000098. asert.err
test000098.ktest
test000098.pc
```

```
objeto 0: tamaño: 29
objeto 0: datos: 'sddwddddssssddwwww \ x00 \ x00 \ x00 \ x00 \
x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00'
```

PRUEBA DE PROGRAMAS RUST CON KLEE

El compilador de Rust se basa en la plataforma LLVM y KLEE se basa en LLVM, por lo que los pasos necesarios para utilizar KLEE para verificar los programas de Rust son:
Compile el programa para generar código de bits LLVM.
Ejecute KLEE en el archivo de código de bits.
Examine la salida de KLEE.
Un pequeño programa de prueba
Como ejemplo continuo, usaremos el mismo ejemplo que usamos para explicar cómo usar la caja de anotaciones de verificación .
Este código está en demos/simple/kleey los comandos de shell en este archivo están en demos/simple/klee/verify.sh.
utilice las anotaciones_de_verificación como verificador;

```
fn  principal () {
     dejar un: U32  = verificador :: AbstractValue :: abstract_value ();
    deje b: u32  = verificador :: AbstractValue :: abstract_value ();
    verificador :: asumir ( 1  <= a && a <=  1000 );
    verificador :: asumir ( 1  <= b && b <=  1000 );
    Si verificador :: is_replay () {
         eprintln! ( "Valores de prueba: a = {}, b = {}" , a, b);
    }
    sea r = a * b;
    verificador :: ¡afirmar! ( 1  <= r && r <=  1000000 );
}
```

El compilador de Rust y KLEE están en el Dockerfile (ver instalación), así que inicie la imagen de Docker ejecutando
demostraciones en cd / simple / klee
Docker / ejecutar
Todos los comandos restantes en este archivo se ejecutarán en esta imagen acoplable.

KLEE: DISPONIBLE GRATUITAMENTE COMO CÓDIGO ABIERTO

- Popular herramienta de ejecución simbólica con una base de usuarios y desarrolladores activos
- Ampliada de muchas formas interesantes por varios grupos de la academia y la industria, en áreas como:
- generación de exploits; Si una vulnerabilidad es la ventana abierta en el sistema, un exploit es la cuerda o la escalera que el ladrón utiliza para llegar a la ventana abierta. Un exploit es simplemente una herramienta creada para aprovechar una vulnerabilidad concreta: si no hay vulnerabilidades, no hay nada que explotar.
- redes de sensores inalámbricos / sistemas distribuidos
- depuración automatizada
- verificación del comportamiento del cliente en línea juegos
- pruebas y verificación de GPU • etc. etc.

BIBLIOGRAFÍA

- Option 1 Página principal de klee- http://klee.doc.ic.ac.uk/#
- Option 2 Página con todos los tutoriales de klee - http://klee.github.io/tutorials/
- Option 3 Documentación de klee https://www.usenix.org/legacy/event/osdi08/tech/full_papers/cadar/cadar.pdf

GRACIAS