6.1: Korat

La primera de las herramientas que vamos a tratar en este apartado se trata de Korat, una extensión de Java que sirve para la generación de casos complejos de prueba a partir de unas restricciones dadas.

La idea detras de Korat es que dado un predicado en Java y una funcion **finilization** en la cual definimos los dominios para cada uno de las clases del input, es decir los valores válidos para cada una de ellas. Korat genera un esqueleto básico de esta función por si solo que puede ser modificado despues por el programador para cambiar los dominios.

Espacio de estados

Lo primero que hace Korat es reservar el espacio necesario para los objetos especificados por ejemplo en el caso de un BinTree reservaria espacio para un BinTree y para el número de Nodos que queramos. Por ejemplo si queremos un arbol con tres nodos el vector contendría 8 campos: * 2 para el BinTree (uno para la raíz y otro para el tamaño) * 2 campos por cada uno de los 3 nodos (hijo izquierdo/hijo derecho)

Cada uno de los posibles candidatos que baraje Korat a partir de ese momento será una evaluación de esos 8 campos. Por lo tanto el espacio de estados de busqueda del input consiste en todas las posibles combinaciones de esos campos, donde cada uno de ellos toma valores de su dominio definido en **finizialization**

Proceso de busqueda

Para conseguir explorar de manera sistemática y completa el espacio de estados Korat ordena todos los elementos en los dominios de las clases y los dominios de los campos. Dicho orden de cada uno de los dominios de los campos será consistente con el orden del dominio de la clase y todos los valores que pertenezcan al mismo dominio de clase ocurriran de manera consecutiva en el dominio del campo.

Cada uno de los candidatos de la entrada es un vector de indices de su correspondiente dominio del campo. Teniendo en cuenta que el dominio de la clase Nodo en el anterior ejemplo cuenta con 3 elementos [N0,N1,N2] a estos debemos añadir null obteniendo asi un dominio de campo [null, N0, N1, N2] que será el dominio de los campos raiz e hijos derecho e izquierdo de cada uno de los nodos. El dominio del campo tamaño será un único entero, 3.

Tras definir los dominios de cada uno de los campos del vector la busqueda comienza con la inicialización de todos los indices del vector a cero. Tras ello para cada posible candidato fijamos los valores de los campos de acuerdo a los valores en el vector y acto seguido invoca a la funcion repOk que es donde el usuario ha definido la precondición de la función. Durante dicha ejecución Korat

monitoriza en que orden son accedidos los campos del vector durante dicha ejecución y construye una lista de los identificadores de los campos, ordenados por la primera vez que repOk los accede.

Cuando repOk retorna Korat genera el siguiente candidato incrementando el indice del dominio de campo para el campo que se encuentra último en la lista ordenada construida previamente. Si dicho indice es mayor que el tamaño del dominio de su campo este se pone a cero y se incrementa el indice de la posición anterior y asi repetidamente.

Al seguir este método para generar el siguiente candidato conseguiremos podar un gran numero de los candidatos que tienen la misma evaluación parcial. Asimismo podemos estar seguros de que dicha poda no deja fuera ninguna estructura de datos válida porque repOk no leyó dichos campos y podría haber devuelto falso independientemente de su valor. Gracias a esta poda de una gran parte del espacio de busqueda Korat puede explorar espacios de busqueda muy grandes de manera eficiente, dicha eficiencia de la poda depende del método repOk por ejemplo si siempre lee todos los datos de entrada antes de terminar la ejecución forzará a Korat a explorar casi todo el espacio de busqueda sin apenas podas.

El algoritmo de busqueda descrito aquí genera las entradas en orden lexicográfico. Ademas para los casos en los que repOk no es determinista este método garantiza que todos los candidatos para los que repOk devuelve true son generados, los casos para los que repOk siempre devuelve false nunca son generados y los casos para los que alguna vez se devuelve true y a veces false pueden ser o no generados.

Resultados no-isomorfos

Dos candidatos se definen como isomorfos si las partes de sus grafos alcanzables desde la raiz son isomorfas. En el caso de repOk el objeto raiz es el objeto pasado como argumento implicito.

El isomorfismo entre candidatos divide el espacio de estados en particiones isomórficas (debido al ordenamiento lexicográfico introducido por el orden de los valores de los dominios de los campos y la ordenación de los campos realizado por repOk). Para cada una de dichas particiones isomomorficas Korat genera únicamente el candidato lexicográficamente menor.

Ademas con el proceso explicado anteriormente para ir generando el siguiente candidato teniendo en cuenta la lista de ordenación de los campos Korat se asegura de no generar varios candidatos dentro de la misma partición isomórfica.