(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 563 487**

(51) Int. CI.:

G06F 9/45 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **19.04.2002 E 02008868 (8)**

(97) Fecha y número de publicación de la concesión europea: **20.01.2016 EP 1258805**

(54) Título: **Colocar instrucción de lanzamiento de excepción en código compilado**

(30) Prioridad:

14.05.2001 US 855239

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
15.03.2016

(73) Titular/es:

MICROSOFT TECHNOLOGY LICENSING, LLC (100.0%)
One Microsoft Way
Redmond, WA 98052, US

(72) Inventor/es:

BHANSALI, SANJAY;
DASAN, SHAJAN;
BRIAN, HARRY D. y
MORRISON, VANCE P.

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Colocar instrucción de lanzamiento de excepción en código compilado

Campo técnico

La presente invención se refiere a compiladores, y más particularmente, al manejo de excepciones de compilación.

Antecedentes y sumario

En general, los compiladores traducen el código fuente legible humano a código objeto legible por máquina en preparación para ejecución. Un compilador está diseñado para recibir código fuente legible humano en un formato esperado. Cuando el código fuente se escribe en un formato no esperado por el compilador, entonces la compilación se aborta y se genera una excepción para identificar el formato inesperado. Existen muchas causas de errores de compilación conocidas en la técnica. Si la compilación se aborta, el código objeto no está disponible para ejecutarse.

Tradicionalmente, el código fuente se compila durante la etapa de desarrollo de software cuando los desarrolladores de software aún están presentes para reparar cualquier error en el programa. Una vez que se reparan los errores, el código fuente se compila en código objeto (conocido también como código nativo o código máquina) para una plataforma objetivo y se prueba. Después de probar, el código objeto puede utilizarse mediante cualquier sistema compatible. Los desarrolladores de software a menudo desarrollan código objeto para diferentes sistemas y plataformas.

Comúnmente se usa un número de términos en la técnica informática para describir el proceso de transformar un programa informático en un formato usable por máquina. Los siguientes términos presentan una vista global amplia y general de los métodos conocidos en la técnica para realizar esta transformación.

Compiladores. Los compiladores de lenguaje de alto nivel toman el código fuente legible humano y lo traducen a código objeto legible por máquina. Tradicionalmente, la compilación de todo el programa se completa antes de que se ejecute cualquier porción del programa. Un compilador a menudo toma código fuente legible humano como entrada y crea código ejecutable por máquina.

Intérpretes. Generalmente, un intérprete es un programa que traduce un programa de un lenguaje a otro y a continuación típicamente poco tiempo después ejecuta el código traducido. En algunos casos el código traducido por el intérprete no se almacena en caché (o se graba en memoria). En estos casos, si una sección de código ya ejecutada necesita ejecutarse de nuevo, debe reinterpretarse.

Máquinas virtuales. La expresión máquina virtual conlleva varios significados. Máquina virtual se usa para describir un entorno informático en tiempo de ejecución que posibilita que se ejecute un programa en un entorno informático distinto al entorno informático para el que se diseñó originalmente el programa. Por ejemplo, un servidor de sistema operativo de 32 bits podría presentar un entorno de máquina virtual en tiempo de ejecución para ejecutar aplicaciones de 16 bits no nativas. La expresión máquina virtual se usa también para describir el entorno de tiempo de ejecución de Java. El entorno de tiempo de ejecución de Java está basado en un lenguaje modelo anterior denominado UCSD Pascal. UCSD Pascal toma código fuente Pascal y lo traduce a un código de lenguaje intermedio que puede interpretarse además (traducirse) en tiempo de ejecución a código objeto y ejecutarse en el entorno nativo. El código de lenguaje intermedio es usable mediante cualquier ordenador con un intérprete que pueda traducir el código de lenguaje intermedio a un código objeto nativo. Un intérprete de este tipo se denomina en ocasiones una máquina virtual.

Compiladores Justo en el momento (JIT). Tal como un compilador tradicional, un compilador JIT traduce un programa desde un primer lenguaje a un segundo lenguaje. Sin embargo, un compilador JIT a menudo retrasa la traducción hasta que la sección o el método del programa en el primer lenguaje son necesarios para la ejecución en un segundo lenguaje. La decisión de cuándo traducir exactamente una sección de código desde un primer lenguaje a un segundo lenguaje puede variar ampliamente. Preferentemente, el código se traducirá en el momento para el que pueda consumirse mediante el programa o el procesador que espera el código en el segundo lenguaje.

La expresión compilador JIT se usa generalmente en el contexto de un proceso de traducción de dos fases. La primera fase de traducción empieza con una representación de lenguaje inicial y lo traduce a una representación de lenguaje intermedio. La segunda fase de traducción empieza con una representación de lenguaje intermedio y lo traduce a una representación de lenguaje posterior. A menudo, el compilador JIT está asociado con la segunda fase de traducción donde el código de lenguaje intermedio se traduce a una representación de lenguaje máquina más amigable. Si la representación de lenguaje máquina amigable es código objeto, entonces el código objeto se usará directamente mediante el procesador. Sin embargo, si la salida del compilador JIT es un código de lenguaje intermedio, debe traducirse además (por ejemplo, mediante una máquina virtual o un intérprete) antes de que pueda utilizarse mediante el sistema nativo. A menudo, los compiladores JIT traducen código de lenguaje intermedio a código nativo antes, y después almacenan (o almacenan en caché) el código traducido en caso de que las secciones de código se ejecuten múltiples veces.

Traductores. Un traductor es generalmente un programa que traduce un programa de un lenguaje a otro. La segunda representación de lenguaje del programa puede ser cualquiera de un lenguaje intermedio o código máquina. La traducción se utiliza generalmente para transformar un programa a lo largo del espectro a partir de un formato legible más humano en un formato máquina más amigable. Para esta memoria descriptiva, los compiladores, compiladores JIT, traductores, intérpretes, ensambladores, máquinas virtuales y cualquier otro programa automatizado y/o proceso que traduzca desde una primera representación de lenguaje a una segunda representación de lenguaje se denominará un traductor.

El proceso de traducción es típicamente tan complejo que a menudo se descompone lógicamente (y potencialmente físicamente) en una serie de sub-unidades. Una lista parcial de tales sub-unidades lógicas incluiría análisis léxico, análisis sintáctico, creación de código intermedio, optimización de código, generación de código, gestión de tablas, comprobación de tipos, verificación de código y manejo de errores. Una enumeración más completa de las sub-unidades de traducción potenciales es bien conocida en la técnica. Para cada una de las sub-unidades potenciales empleadas para traducir un programa desde un primer lenguaje (por ejemplo, código fuente o código intermedio) a un segundo lenguaje (por ejemplo, código intermedio, código objeto o código interpretable), algo puede ir mal con el proceso. Cuando una sub-unidad de traducción aborta la traducción puesto que no puede resolver el significado de una instrucción o instrucciones, entonces esa instrucción se considera como instrucción o instrucciones irresolubles (o código irresoluble). (Véase a continuación Errores de Traducción Irresolubles). Si el traductor no puede resolver la traducción de una manera significativa o no ambigua, entonces la traducción se aborta.

Por muchas razones incluyendo la interoperabilidad y la seguridad, el código de lenguaje intermedio ha estado ganando popularidad. El código de lenguaje intermedio es una representación de un programa que se tradujo desde una representación de código de lenguaje inicial del programa. Además, la representación del código de lenguaje intermedio del programa se traducirá a una representación de código de lenguaje posterior del programa algún tiempo antes de que el programa se ejecute. A menudo, el código fuente se traduce a un código de lenguaje intermedio en una primera traducción. Este código de lenguaje intermedio se traduce a continuación a un código objeto nativo en una segunda traducción. La primera y segunda traducciones pueden tener lugar en el mismo o en diferentes ordenadores. Potencialmente, un programa puede traducirse a representaciones de código de lenguaje intermedio diferentes en múltiples fases de representación intermedias sucesivas. Cualquier ordenador que tenga un traductor que pueda traducir alguna fase de código de lenguaje intermedio a un código de lenguaje posterior utilizable, puede usar la funcionalidad del programa representado mediante el código de lenguaje intermedio.

Los desarrolladores pueden crear un programa de código fuente antes, traducirlo a código de lenguaje intermedio y distribuirlo a cualquier plataforma con un traductor de código de lenguaje intermedio. Un código de lenguaje intermedio universal es eficaz puesto que es más fácil para cada ordenador soportar uno o dos (o unos pocos) traductores de código de lenguaje intermedio que soportar múltiples compiladores de código fuente diferentes (tradicionales). Además, en el momento que un programa alcanza una fase de código de lenguaje intermedio, mucha de la dificultad o tiempo que consume el análisis puede ya haberse calculado mediante una fase de traducción anterior, reduciendo de esta manera la carga de trabajo para un traductor posterior.

Sin embargo, la traducción de múltiples etapas crea un nuevo conjunto de problemas potenciales. En traducciones posteriores, si un siguiente traductor lanza una excepción de traducción, los desarrolladores de código fuente pueden ya no estar presentes para corregir algún error de traducción. Por lo que si el traductor de lenguaje intermedio aborta la traducción, el programa representado mediante el código de lenguaje intermedio no será usable. El código de lenguaje intermedio es inservible incluso si la sección de instrucción o instrucciones irresolubles que produjeron que se abortara la traducción nunca se hubieran ejecutado en un caso específico después de la traducción. No es inusual que no se ejecuten grandes porciones de un programa informático durante algún caso de ejecución individual.

T. Lindholm y col.: “The Java™ Virtual Machine Specification”, septiembre de 1996, Addison-Wesley, pertenece a los libros de la Serie de Java que proporcionan documentación de referencia para programadores y usuarios finales de Java. La especificación describe la Versión 1.0.2 de la Máquina Virtual de Java, en la cual se proporciona una vista global de conceptos de Java y de la Máquina Virtual de Java. La especificación define también el formato de fichero de clase junto con un formato de fichero independiente de plataforma y de implementación para código Java compilado y describe la gestión de tiempo de ejecución del grupo de constantes así como el conjunto de instrucciones de la Máquina Virtual de Java. Se proporcionan ejemplos de código de Java de compilación en el conjunto de instrucciones de la Máquina Virtual de Java o de una optimización y se describen subprocesos de la Máquina Virtual de Java y su interacción con memoria.

Sumario de la invención

La invención es lo que se especifica en las reivindicaciones independientes.

Se especifican realizaciones preferidas en las reivindicaciones dependientes.

La presente invención se refiere a completar la traducción cuando se encuentra código de entrada irresoluble durante la traducción en lugar de abortar. Esto puede conseguirse insertando una instrucción o instrucciones de

lanzamiento de excepción (y/o instrucción o instrucciones de manejo) en el código de salida donde se hubiera colocado el código de entrada irresoluble en el código de salida si hubiera sido traducible. En una realización de este tipo, tras encontrar una instrucción o instrucciones irresolubles en el código de entrada, el traductor continúa la traducción y coloca una instrucción de lanzamiento de excepción en el código de salida. Una instrucción de lanzamiento de excepción insertada de este tipo se lanzará si alguna vez se ejecuta.

Esta memoria descriptiva analiza tres tipos generales de instrucciones o código de entrada: instrucción o instrucciones (o código) traducibles, instrucción o instrucciones (o código) irresolubles, e instrucción o instrucciones (o código) dudosas. Las instrucciones traducibles se traducen por consiguiente al código de salida. Las instrucciones irresolubles no se traducen puesto que su significado pretendido no es conocido, por lo que se convierten en su lugar en una instrucción o instrucciones de lanzamiento de excepción (y/o instrucciones de manejo) y se insertan o colocan en el código de salida. Esta inserción o reemplazo de código irresoluble con otras instrucciones se denomina también reemplazo. Una instrucción o instrucciones dudosas es código de entrada que es literalmente traducible, pero por alguna otra razón, permanece ambiguo para el traductor. Un código dudoso de ejemplo trivial es una instrucción que es sintácticamente correcta pero contiene una asignación a una variable que no puede comprobarse con seguridad el tipo.

En una realización, el traductor traducirá la instrucción dudosa al significado ambiguo al código de salida, con la condición de que la instrucción dudosa sea desde una fuente confiable. En una realización de este tipo, si la instrucción no es desde una fuente confiable, la instrucción dudosa se reemplazará con una instrucción de lanzamiento de excepción (y/o instrucción de manejo) en el código de salida.

En una realización, la instrucción de lanzamiento de excepción (y/o instrucción de manejo) pueden insertarse en el código de salida en un número de localizaciones potenciales. La instrucción puede insertarse en el lugar o delante de la propia instrucción o instrucciones irresolubles, el bloque básico, un procedimiento, un método, un objeto o un módulo que contiene la instrucción o instrucciones irresolubles, o en lugar de o delante de una llamada o ramificación a un procedimiento, método, objeto o módulo que contiene la instrucción o instrucciones irresolubles. Además, la instrucción o instrucciones pueden insertarse en lugar de o delante de una llamada (en tiempo de traducción o en tiempo de ejecución) a un servidor local o remoto enlazado o biblioteca que contiene una instrucción o instrucciones irresolubles. Además, la instrucción puede insertarse en cualquier lugar en la ruta de ejecución en o delante de donde hubiera aparecido la instrucción irresoluble si hubiera sido traducible. Finalmente, puede insertarse un nuevo bloque básico que contiene las instrucciones de lanzamiento de excepción y/o manejo en el código traducido, con el flujo de control dirigido al nuevo bloque básico en el caso de que la ruta de ejecución alcanzara de otra manera una instrucción o instrucciones irresolubles u otra condición o condiciones intraducibles.

En otra realización, el traductor no aborta la traducción cuando alcanza un bloque básico que contiene una instrucción o instrucciones irresolubles. En su lugar, el traductor inserta una instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) como la primera instrucción en un bloque básico en el código traducido. La excepción se lanza si y cuando ese bloque básico se ejecuta alguna vez.

En otra realización, el traductor traducirá el código dudoso sin colocar una instrucción de lanzamiento de excepción (y/o instrucción de manejo) en el código traducido, siempre que el código dudoso esté dentro de una clase definida de código dudoso. Por ejemplo, una clase definida de código dudoso se define como código que es dudoso basándose en una incapacidad de probar la seguridad del tipo del código en casos cuando el código dudoso es desde una fuente confiable (por ejemplo, desarrollador de software, distribuidor, etc.).

En otra realización, los metadatos en el fichero de entrada identifican la fuente de entrada. El traductor considera la fuente del flujo de entrada como se describe o identifica en los metadatos para determinar si eliminar una instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) que se insertarían de otra manera debido a que el traductor (por ejemplo) no puede comprobar el tipo.

En una realización, los metadatos en el flujo de entrada recomiendan el nivel (por ejemplo, nivel de método, nivel de instrucción, nivel de bloque básico, nuevo bloque básico, etc.), en que debería insertarse la instrucción o instrucciones de lanzamiento de excepción en el flujo de salida.

En otra realización, una excepción convencional (por ejemplo, x86) se inserta para todos los casos cuando se inserta una instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) en el código traducido. En otra realización, los metadatos en el fichero de entrada recomiendan clases de instrucción o instrucciones de lanzamiento de excepción potenciales (y/o instrucción o instrucciones de manejo) para insertar en el código traducido. En otra realización, los metadatos recomiendan una rutina de manejo de excepción para manejar una excepción insertada cuando se lanza. Una realización de este tipo considera las clases recomendadas de manejadores de excepción para insertar en el flujo de salida. En otra realización, los metadatos sugieren manejadores de excepción para áreas correspondientes (instrucciones, bloque básico, procedimiento, objeto,...) del flujo de entrada para casos cuando se encuentran instrucción o instrucciones irresolubles en tales áreas.

En otra realización, una copia no modificada del código irresoluble se incluye en el flujo de salida con las instrucciones de lanzamiento de excepción insertadas (y/o instrucción o instrucciones de manejo), y se realiza un

segundo intento de tiempo de ejecución para traducir el código irresoluble. En tiempo de ejecución, si el código irresoluble se traduce, entonces se ejecutará la traducción resultante. Si el código irresoluble permanece intraducible, entonces se ejecutará la instrucción de lanzamiento de excepción insertada (y/o instrucción de manejo).

La realización ilustrada completa la compilación a pesar del código irresoluble colocando una instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) en el código objeto o en el flujo de salida código de lenguaje intermedio. Si la sección de programa que contiene la instrucción o instrucciones de lanzamiento de excepción se ejecuta más tarde, se lanza la excepción. Cómo se maneja esa excepción dependerá del manejo de excepción implementado y si los metadatos contenidos en el flujo de entrada describen o no instrucciones de lanzamiento de excepción o manejadores de excepción potenciales. Además, si una realización se convierte o no en cualquier metadato de fichero de entrada de este tipo en correspondientes instrucciones de lanzamiento de excepción o instrucciones de manejo de excepción en el fichero de salida. Una realización puede tener o no en cuenta alguna recomendación contenida en los metadatos del fichero de entrada de código de lenguaje intermedio. Además, una realización puede o contener o no metadatos que describen manejadores de excepciones o de excepción potenciales en el fichero de entrada.

Finalmente, los errores de traducción irresolubles no están limitados a instrucciones irresolubles. Pueden ser cualquier condición en el flujo de entrada que un traductor determina que requiere instrucción o instrucciones de lanzamiento de excepción y/o manejo en el flujo de salida.

En cualquier realización de este tipo, la instrucción o instrucciones de lanzamiento de excepción insertadas o colocadas (y/o instrucciones de manejo) pueden ser instancias de objetos de excepción (y/o de manejo) seleccionados desde una biblioteca de clases. Además, todas las realizaciones de este tipo pueden enlazarse en tiempo de traducción o en tiempo de ejecución, dinámica o estáticamente, local o remotamente.

Se harán evidentes características y ventajas adicionales a partir de la siguiente descripción detallada de la realización ilustrada que continúa con la referencia a los dibujos adjuntos.

Breve descripción de los dibujos

La Figura 1 es un diagrama de flujo de datos que representa un programa que se traduce a un segundo lenguaje.
La Figura 2 es un diagrama de flujo de datos que representa una traducción fallida y abortada y un listado de errores de traducción.
La Figura 3 es un diagrama de bloques que representa un proceso de traductor que contiene un universo parcial de sub-unidades de traducción lógicas.
La Figura 4 es un diagrama de flujo de datos que representa una traducción continuada tras identificar una entrada irresoluble y una salida resultante que contiene una instrucción de lanzamiento de excepción.
La Figura 5 es un listado de un método.
La Figura 6 es un listado de programa que representa la identificación de bloque básica.
La Figura 7 es un diagrama que representa la identificación de bloque básica en el contexto de una sección de código muerta.
La Figura 8 es un listado de programa que representa colocación potencial de instrucciones de lanzamiento de excepción para métodos o para métodos sin múltiples bloques básicos.
La Figura 9 es un diagrama de flujo de traducción de sub-unidades de traducción.
La Figura 10 es un listado de programa que representa Bloques Básicos, uno de los cuales tiene una instrucción de lanzamiento de excepción insertada, una captura de excepción y un manejador de excepción.
La Figura 11 es un diagrama que representa dos ejemplos separados de ventanas generadas en respuesta a un manejador gráfico.
La Figura 12 es un diagrama que representa un Fichero o Flujo de Entrada.
La Figura 13 es un diagrama que representa Bloques Básicos; conteniendo uno un manejador insertado.
La Figura 14 es un diagrama de bloques de un listado de flujo de salida que incluye una estructura de datos de manejador.
La Figura 15 es un diagrama que representa unas diversas rutas de ejecución a través de diversos bloques básicos, una pila y un bloque de lanzamiento de excepción y/o de manejo insertado.
La Figura 16 es un diagrama que representa operandos insertados en la pila antes de saltar a un bloque objetivo.
La Figura 17 es un diagrama de bloques de un sistema informático distribuido que implementa un método y aparato que incorpora la invención.

Descripción detallada

Traductores

Generalmente, como se muestra en la Figura 1, un traductor es un primer programa 100 que toma un segundo programa 101 escrito en un primer lenguaje 102 como entrada y crea una segunda representación de lenguaje del segundo programa 103 como salida. Si el primer lenguaje es un lenguaje de alto nivel tal como C, C++, Smalltalk o Ada (código fuente), y el segundo lenguaje es un código legible por máquina o código de lenguaje ensamblador, entonces el traductor se denomina un compilador.

Otros traductores convierten desde un lenguaje a otro lenguaje denominado un lenguaje intermedio. A menudo el código de lenguaje intermedio puede ejecutarse directamente mediante un ordenador usando un programa denominado un intérprete. Otras veces el lenguaje intermedio se reduce además a un código objeto antes de que se ejecute. Algunos compiladores de alto nivel usan código de lenguaje ensamblador para un lenguaje intermedio. El código de lenguaje ensamblador es generalmente un código de lenguaje de bajo nivel donde cada instrucción corresponde a una única instrucción de máquina. Los ensambladores son programas que traducen código de lenguaje ensamblador en código máquina. En general, el código objeto es código que es usable directamente mediante la máquina.

Para esta especificación, los compiladores, compiladores JIT, traductores, intérpretes, ensambladores, máquinas virtuales y cualquier otro programa que traduzca desde un primer lenguaje a un segundo lenguaje se denominarán traductores. Estos traductores se definen como tal si representan una única traducción antes de ejecución, o tiene lugar una de múltiples traducciones antes de la ejecución. Además, compilación y compilar, traducción y traducir, interpretación e interpretar y ensamblaje y ensamblar se denominarán traducción o traducir. Dadas tales definiciones, un traductor traduce desde código fuente a código intermedio, desde código intermedio a código objeto, desde un código de lenguaje intermedio inicial en un código de lenguaje intermedio posterior, o desde código fuente a código objeto. El proceso se denominaría traducción o traducir, y tras la finalización de la traducción, el código resultante se denominaría código traducido, flujo de salida o fichero de salida. Puesto que la invención se aplica a todas estas formas de traducción, usamos traducción en este sentido general.

Generalmente, los traductores 100 están diseñados para reconocer un fichero de código en un primer lenguaje (fichero de entrada o flujo de entrada) 102 y convertirlo en un fichero de código en un segundo lenguaje (fichero de salida o flujo de salida) 103. Cuando alguna porción del fichero de entrada en el primer lenguaje es intraducible por alguna razón, históricamente el traductor se detiene y la traducción falla al crear el fichero de salida en el segundo lenguaje. A menudo, el traductor notifica al usuario del intento de traducción fallido y enumera las líneas de código que contienen los errores de traducción irresolubles.

Errores de traducción irresolubles

Como se muestra en la Figura 2, lo que todos estos traductores 200 tienen en común es un potencial para identificar instrucciones, datos u objetos 204 en el flujo de entrada (en esta memoria descriptiva “fichero de entrada” tiene el mismo significado efectivo que “flujo de entrada” y “fichero de salida” tiene el mismo significado efectivo que “flujo de salida”) del primer lenguaje que por alguna razón u otra el traductor no puede traducir inequívocamente, significativamente o de manera segura en instrucciones, datos u objetos correspondientes en el segundo lenguaje 202, 208. En un caso de este tipo, el traductor típicamente abortará la traducción y suministrará un mensaje 206 a un dispositivo de salida (pantalla, fichero o impresora) 203 que identifica la fuente de la instrucción o instrucciones, datos u objetos irresolubles en el flujo de entrada 204.

Los tipos de errores de traducción encontrados son bastante diversos y se analizan en la técnica anterior. Históricamente, debido a que el tiempo de compilación siempre precedía al tiempo de ejecución, estos errores se denominaban errores de compilación. Más generalmente, cada vez que un traductor no puede traducir inequívocamente, significativamente o de manera segura una porción de código en un programa representado en un primer lenguaje en un programa representado en un segundo lenguaje, se denomina “código irresoluble”, “errores de traducción irresolubles,” “errores de traducción” o “código defectuoso”. Tales errores de traducción irresolubles no están limitados. Pueden ser cualquier condición en el flujo de entrada que un traductor determina que requiere instrucción o instrucciones de lanzamiento de excepción y/o manejo en el flujo de salida.

Cuando tal código 204 irresoluble se encuentra en el flujo de entrada del primer código de lenguaje durante la traducción, se genera y entrega un mensaje de error de traducción a un dispositivo 203 de salida. A menudo la traducción continuará y el traductor generará una lista de errores de traducción al dispositivo 207 de salida. Esto permitirá que se corrijan múltiples errores mediante el desarrollador de software antes de intentar la traducción de nuevo.

El número y tipos de errores de traducción que existen en la técnica de traducción de lenguaje informático automatizado (por ejemplo, compiladores, intérpretes, traductores, intérpretes, máquinas virtuales, ensambladores, enlazadores, y etc.) son muy diversos. Los siguientes errores ilustran únicamente un sub-conjunto potencial del universo de errores que pueden tener lugar durante la traducción y la expresión “error de traducción” debe abarcar el significado más amplio. Un entendimiento más diverso de errores de traducción de compilador, traductor, intérprete, ensamblador o máquina virtual es conocido en la técnica informática. Véase Alfred V. Aho, Compilers, Principles, Techniques, and Tools Addison-Wesley Publishing Co., 1986, y véase Reinhard Wilhelm, Compiler Design, Addison-Wesley Publishing Co., 1995.

Pueden surgir errores de traducción durante cualquiera de las sub-unidades lógicas de traducción. En el traductor 300 de ejemplo mostrado en la Figura 3, se representa 301-314 una lista parcial de las sub-unidades lógicas potenciales. Cada traductor tendrá una lista potencialmente diferente de tales sub-unidades lógicas y muchos traductores tendrán diferentes nombres para sub-unidades que realizan funciones similares. Durante la traducción en una cualquiera de estas sub-unidades lógicas, puede generarse un error de traducción.

Un error de traducción de sintaxis puede tener lugar durante el análisis 302 de sintaxis cuando falta un punto y coma del final de una declaración, o a una expresión aritmética le falta uno o más paréntesis. Un error de traducción semántica puede tener lugar durante el análisis 303 semántico cuando un operando aritmético se aplica a un testigo textual, o un entero se asigna a un identificador definido para apuntar a un objeto de estructura de datos. Un error de traducción léxico puede tener lugar durante el análisis 301 léxico cuando se escribe mal una palabra clave, o cuando se hace referencia a un identificador indefinido. Un error de traducción lógico puede tener lugar durante el análisis 311 lógico por un fallo de diseño pasado por alto. Un error de traducción de flujo de entrada puede producirse por una degradación de la señal evidenciada por bits perdidos en el flujo de entrada haciendo de esta manera el flujo de entrada irreconocible. Un error de traducción de verificación de código puede tener lugar durante el análisis 310 de verificador cuando el traductor no puede resolver de manera significativa una porción del código de entrada. Véase a continuación “Verificación de Código”. Un error de traducción de comprobación de tipo puede tener lugar cuando se asigna una variable de número real a un índice de serie.

Cuando tienen lugar uno o más errores de traducción durante la traducción, la instrucción o instrucciones o sección de código que el traductor no puede traducir se denomina código defectuoso, código irresoluble o errores de traducción irresolubles.

Típicamente, cuando un traductor (o cualquier sub-unidad de traducción) encuentra código irresoluble, el proceso de traducción puede continuar para los fines de señalar otro código irresoluble más adelante en el flujo de entrada. Identificando múltiples errores en el fichero de entrada, el desarrollador de software puede resolver múltiples problemas antes de intentar traducir el programa de nuevo. Sin embargo, como se muestra en la Figura 2, un fichero de salida que contiene el programa representado en el segundo lenguaje no se devuelve para ejecución 208. En un intento de traducción fallido, la salida desde el traductor está limitada más generalmente a un listado y posible descripción de los errores 203, 207 de traducción.

En una realización de la invención, como se muestra en la Figura 4, la traducción continúa tras identificar el código 404 irresoluble y se genera 408 un flujo (o fichero) de salida que contiene el código traducido, y se coloca una instrucción de lanzamiento de excepción en el flujo de salida en lugar del código 409 irresoluble. En otra realización, la instrucción de lanzamiento de excepción puede colocarse en el segundo flujo de salida de lenguaje en lugar del código irresoluble si ese segundo flujo de salida de lenguaje es (i) el flujo de salida de la primera traducción en una serie de N traducciones, (ii) la última traducción en una serie de N traducciones, o (iii) cualquier traducción en la serie de traducciones una a N, donde N es mayor que o igual a uno.

Colocar instrucciones de lanzamiento de excepción en código compilado

En una realización de la invención, un programa representado en un primer lenguaje 401 se traduce a una representación del programa en el segundo lenguaje 408 incluso aunque una porción del código sea irresoluble 404. La porción irresoluble y potencialmente otras instrucciones en las proximidades de la porción irresoluble se reemplazan con la instrucción o instrucciones 409 de lanzamiento de excepción. En el caso de que estas instrucciones de lanzamiento de excepción se ejecuten alguna vez, se lanzará la excepción.

Una realización de este tipo permite la finalización de la traducción desde un primer lenguaje a un segundo lenguaje 408. Además, parece razonable esperar que el código irresoluble reemplazado con una instrucción de lanzamiento de excepción, nunca pueda ejecutarse. En una realización de este tipo, la excepción se lanzará únicamente si y cuando se ejecute la instrucción o instrucciones que reemplazaron (o se insertaron delante de) el código irresoluble. Puesto que la traducción del programa se completa desde el primer lenguaje al segundo lenguaje en esta realización, el programa está disponible en el segundo código de lenguaje para utilización (por ejemplo, ejecución, traducción, interpretación, compilación, compilación JIT, ensamblaje).

Hay muchos casos donde puede ser beneficioso completar la traducción desde un primer lenguaje en un segundo lenguaje, donde el segundo lenguaje contiene una instrucción de lanzamiento de excepción insertada en el código traducido en lugar de simplemente abortar la traducción y producir una lista de errores de traducción. Por ejemplo, a menudo un desarrollador de software está usando ficheros de clases de terceros que contienen clases enumeradas junto con una definición de interfaz para utilizar tales clases. No es extraño al enlazar el código de los desarrolladores de software a tales bibliotecas de clase para que se encuentre un error de traducción. Sin embargo, a menudo el desarrollador de software no tiene el código fuente para las bibliotecas de clase y posteriormente no puede completar la traducción para obtener código ejecutable. Los desarrolladores de software pueden preferir completar la traducción y obtener código traducido que contiene la instrucción o instrucciones de lanzamiento de excepción de modo que pueden probarse porciones del diseño de software.

Otro ejemplo de cuándo una traducción completada puede ser deseable es en el entorno de internet. A menudo, el código de lenguaje intermedio se suministra a través de la red con la espera de que se traducirá desde un lenguaje de plataforma neutral (primer lenguaje) en un lenguaje específico de plataforma (segundo lenguaje) para utilización en el ordenador local. En un caso de este tipo, si la traducción se completa a pesar del código irresoluble, el ordenador local puede utilizar el código traducido y ejecutará únicamente la instrucción de lanzamiento de excepción si y cuando se ejecute la sección de programa que contuvo antes una instrucción irresoluble. Permitir la traducción en un entorno de red de este tipo es útil puesto que es más probable que el código de lenguaje intermedio (primer

flujo de entrada de lenguaje) no será en general tan amigable para el humano. En un caso de este tipo, el código irresoluble es más difícil de detectar, más difícil de reparar y es más probable que un usuario de internet típico no pueda descubrir y reparar (depurar) el código irresoluble en el flujo de entrada de código de lenguaje intermedio.

Otro ejemplo es una instrucción en el primer código de lenguaje que solicita un enlace a un recurso a partir del cual alguna porción del programa que se está traduciendo solicitará servicios o buscará definición. Si el enlace de tiempo de traducción no puede realizarse por alguna razón (por ejemplo, la ruta del enlace es errónea, el enlace ya no existe, la ruta de red está sin funcionamiento, el servidor de enlace sin funcionamiento, etc.), entonces una realización de la invención coloca una instrucción de lanzamiento de excepción en el lugar de la instrucción que solicita el recurso enlazado. El enlace puede ser un enlace de tiempo de traducción, que enlaza módulos ejecutables en el segundo flujo de salida de lenguaje, o un enlace de tiempo de ejecución, que enlaza el flujo de salida de ejecución a una biblioteca dinámicamente enlazable local o remota de módulos ejecutables. En un caso de este tipo, el enlace a menudo facilita servicios que no se utilizarán en cada caso de la ejecución de flujo de salida, y crear el código traducido que contiene la instrucción de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) en lugar de la instrucción o instrucciones irresolubles puede ser una alternativa viable a un fallo de traducción.

Además, en otra realización, si el enlace solicitado es para (o el flujo de entrada que contiene el enlace solicitado es desde) una fuente conocida o confiable, entonces el enlace solicitado puede traducirse en el segundo flujo de salida de lenguaje sin colocar una instrucción de lanzamiento de excepción en el flujo de salida. En ese caso, el enlace inverificable se traduce sin limitación adicional. (Identificar fuentes confiables se analiza a continuación en “Metadatos y Seguridad”.)

También, si el enlace no puede realizarse en tiempo de traducción, pueden colocarse instrucciones de manejo ejecutables en el flujo de salida para intentar enlazar de nuevo en tiempo de ejecución.

Traductores JIT

Para fines de ilustración, y no para limitar el alcance de la invención, se considera un sub-conjunto del universo de problemas potenciales tratados mediante una realización enumerada. Este sub-conjunto considerado son inconsistencias de programa producidas por la dependencia de versión de implementación del traductor.

En el contexto de compilador (o traductor) JIT, el código intermedio a menudo se traduce únicamente momentos antes de que se ejecute. Variando con cuánta antelación se traduce el código de lenguaje intermedio en código objeto mediante un traductor JIT, tendrá un efecto sobre si y cuándo se generan errores de traducción. Puesto que generalmente un programa que manifiesta un resultado inesperado o impredecible es indeseable, sería preferente reducir tal incertidumbre. Los siguientes ejemplos muestran cómo una realización ilustrada reduciría la incertidumbre en el contexto de traductores JIT.

Una primera implementación de versión de traductor espera hasta el último instante para traducir el código intermedio. En ese caso, únicamente intenta traducir código que se ejecutará más tarde. En este primer traductor, no hay riesgo de fallo de traducción innecesaria, puesto que únicamente se traduce código necesario. El error de traducción fue inevitable. Sin embargo, una implementación de versión de traductor posterior puede traducir con mayor antelación siempre que una CPU tenga segmentos de tiempo disponibles. Si la implementación de versión de traductor posterior intenta traducir una sección de código irresoluble que no se ejecuta posteriormente, entonces la traducción se aborta, incluso aunque la sección que contiene el código irresoluble nunca se hubiera ejecutado. Puesto que el primer traductor completó la ejecución mientras que el segundo traductor abortó la traducción, el comportamiento del programa puede variar basándose en la versión de la implementación del traductor. Por lo tanto, el comportamiento del programa será inconsistente e impredecible. Esta incertidumbre puede reducirse con una realización ilustrada.

En una realización, independientemente de con cuánta antelación se traduzca el código de lenguaje intermedio, se coloca una instrucción de lanzamiento de excepción en el flujo de salida en respuesta a identificar la instrucción o instrucciones irresolubles. En una realización de este tipo, una excepción se lanza predeciblemente únicamente si y cuando la instrucción o instrucciones de lanzamiento de excepción insertadas se ejecutan realmente. Esto reduce la imprevisibilidad basándose en la implementación de versión del traductor.

Otra forma de incertidumbre que puede tratarse es la incertidumbre creada por variaciones en el nivel de “inteligencia” de implementaciones de traductor. El comportamiento de programa puede variarse basándose en el nivel de inteligencia (o previsión) de los diseñadores de una implementación de versión de traductor dada. Esto puede ilustrarse usando la siguiente sección de código.

if (x = true) then
“ejecutar código irresoluble”

Un traductor inteligente puede determinar que la variable Booleana “x” siempre será falsa y por lo tanto el código irresoluble nunca se ejecutará. Puesto que el código irresoluble nunca se ejecutará, un traductor inteligente descarta (o ignora) el código irresoluble y traduce el saldo del programa. Sin embargo, un traductor no tan inteligente puede fallar al realizar esta determinación y abortará la traducción tras encontrar el código irresoluble. El traductor inteligente crea código objeto y el traductor no tan inteligente aborta la traducción tras identificar el código

irresoluble. Por lo tanto, el comportamiento del programa es inconsistente e impredecible basándose en la implementación de versión del traductor.

En una realización, independientemente de la inteligencia de un traductor, se coloca una instrucción o instrucciones de lanzamiento de excepción en el flujo de salida en respuesta a identificar el código irresoluble. En un caso de este tipo, el comportamiento de tiempo de ejecución del código traducido no cambia. Si una implementación de versión de traductor inteligente ignora el código, o una implementación de versión de traductor no tan inteligente genera una instrucción o instrucciones de lanzamiento de excepción, puesto que el código nunca se ejecuta, el comportamiento de la ejecución es predecible. En una realización de este tipo, se lanza una excepción predeciblemente y únicamente si y cuando la instrucción o instrucciones de lanzamiento de excepción insertadas se ejecutan realmente. Una realización ilustrada de este tipo reduce la incertidumbre creada por la implementación de versión del traductor.

Otra inconsistencia tratada mediante una realización ilustrada es en el contexto de pre-JIT. Un pre-JIT tiene lugar cuando grandes porciones (o todo) un flujo de entrada se traduce con antelación. Por ejemplo, si una implementación de traductor de primera versión decide realizar pre-JIT todo el fichero de entrada de código de lenguaje intermedio antes de ejecutar alguna porción del mismo, entonces toda la traducción del programa puede abortarse simplemente porque un método o sección contiene instrucción o instrucciones irresolubles. Dada una primera implementación de versión de este tipo, el programa abortado no está disponible incluso en casos donde la sección o secciones del código irresoluble nunca se hubieran ejecutado en un caso de ejecución dado. Si una implementación de traductor de segunda versión siguió una estrategia JIT más convencional, la segunda implementación de versión no intentaría traducir la sección o método que contiene el código irresoluble, completaría la traducción del programa, y pondría el código traducido disponible para ejecución. Por lo tanto, la primera versión abortaría la traducción, y la segunda versión pondría el código traducido disponible para ejecución.

En la realización ilustrada, independientemente de cuándo se traduzca el flujo de entrada, se coloca una instrucción de lanzamiento de excepción en el flujo de salida en respuesta a identificar el código irresoluble. En una realización de este tipo, una excepción se lanza predeciblemente y solamente si y cuando la instrucción o instrucciones de lanzamiento de excepción insertadas se ejecutan realmente. Esto reduce la incertidumbre basándose en la dependencia de la versión de la implementación del traductor.

En otro ejemplo, dos implementaciones de versión de traductores pueden variar cuándo y si las porciones de un programa (flujo de entrada) se traducen en un entorno de múltiples procesos. El comportamiento del programa producido abortando la compilación mientras se intenta traducir la instrucción o instrucciones irresolubles en un entorno multi-proceso puede variar enormemente. Una porción del programa puede ya haberse ejecutado o estar ejecutándose cuando el traductor comienza a traducir una sección que contiene código irresoluble. La traducción se aborta tras identificar el código irresoluble. Si la sección (instrucción o instrucciones) que provocan el aborto de la traducción nunca se hubieran ejecutado en la ruta de ejecución dada, el comportamiento de tiempo de ejecución del programa sería inconsistente basándose en la implementación de versión del traductor. Por lo tanto, el comportamiento de tiempo de ejecución depende de en qué traductor se implemente.

Las realizaciones ilustradas presentan comportamiento más predecible en el entorno multi-proceso. Incluso si se usa el mismo traductor, un programa puede comportarse de manera diferente en diferentes ejecuciones del programa. Las inconsistencias en un comportamiento de tiempo de ejecución son indeseables. El multi-procesamiento es complejo ya sin lanzar errores de compilación JIT. La complejidad de abortar un compilador JIT puede reducirse colocando un lanzamiento de excepción o instrucción de manejo en un flujo de salida ejecutable. La excepción se lanza a continuación únicamente si y cuando esa sección de código se ejecuta realmente. Si la sección de código se ejecuta, se lanza la excepción insertada. Esto es comportamiento mucho más predecible y mucho más sencillo de diagnosticar que si el compilador JIT lanza una excepción.

Las realizaciones ilustradas presentan el potencial para crear comportamiento de tiempo de ejecución más predecible para los problemas ilustrados --de entornos multi-proceso, dependencia de versión, excepciones condicionales y fallo pre-JIT. Todos estos cuatro problemas pueden tener comportamiento de programa más predecible puesto que el comportamiento de programa se predice no basado en si y cuándo se realiza un intento para traducir código irresoluble, sino basado en si y cuándo se inserta una instrucción de lanzamiento de excepción en respuesta al código irresoluble que se ejecuta más tarde. Únicamente en el momento de ejecutar la instrucción o instrucciones de lanzamiento de excepción insertadas se lanzará la excepción. Esto permite que se complete la traducción y la excepción insertada en lugar del código irresoluble únicamente se lanzará, si la instrucción o instrucciones insertadas se ejecutan (o utilizan). Las excepciones se lanzan en tiempo de ejecución en lugar de en tiempo de traducción, y únicamente cuando la excepción insertada está realmente en la ruta de ejecución.

Identificar áreas de código bloqueado mediante instrucciones de lanzamiento de excepción

Cuando una sección de código en un primer lenguaje contiene código irresoluble, una realización puede determinar si cualquiera del código que rodea el código irresoluble no debería ejecutarse.

Suponiendo que existe un programa en un primer lenguaje y se está traduciendo en un segundo lenguaje. Además,

como se muestra en la Figura 5, suponiendo que todo el programa sea traducible al segundo lenguaje excepto la línea de código irresoluble 501 contenida en una parte de un método 500.

El código irresoluble en este procedimiento es la letra 'o' contenida en la expresión aritmética "z:= (y + (X-20o000)*0,18)". En este ejemplo, puesto que la mayoría de las personas ganan menos de 200.000 $ por año, podría decirse que el procedimiento tiene alguna función limitada. Los primeros cuatro casos calcularán los impuestos basándose en los ingresos, y únicamente el quinto caso contiene (un ejemplo trivial de) código irresoluble.

Este ejemplo es únicamente para fines ilustrativos y el hecho que en este caso el primer lenguaje se represente en una forma legible humana no pretende ser limitante. Si el primer lenguaje se representó (por ejemplo) en un código de lenguaje intermedio menos amigable para el humano, entonces no sería tan evidente cómo corregir el error sencillo (por ejemplo, cambiar 'o' a '0'). Además, si el primer lenguaje se representó en un código de lenguaje intermedio que se había recibido mediante el entorno de traducción desde la red, los humanos que encuentran un error de traducción en el entorno de traducción es probable que no tuvieran una copia del código fuente original (el código fuente en que se escribió originalmente el programa) para corregir el flujo de entrada.

En una realización donde el traductor inserta una instrucción de lanzamiento de excepción en lugar de o delante de código irresoluble, un traductor puede determinar dónde colocar (o insertar) la instrucción de lanzamiento de excepción.

Los siguientes niveles son para fines de ilustración únicamente y no se pretenden para limitar los niveles potenciales o colocación de instrucciones de lanzamiento de excepción. Además, se entiende que la Figura 5 es el flujo de entrada que contiene el código 501 irresoluble, y el flujo de salida que contiene una instrucción de lanzamiento de excepción en cualquier nivel de instrucción, nivel de método, o nivel de bloque básico que aparecería en forma significativamente diferente puesto que estaría en un segundo formato de lenguaje (probablemente un lenguaje intermedio o lenguaje máquina que depende de un traductor dado). Sin embargo, la identificación del nivel de instrucción, el nivel de bloque básico, y el nivel de método en el flujo de entrada es informativa puesto que estos niveles a menudo mantienen las mismas relaciones lógicas en el segundo flujo de salida de lenguaje incluso aunque tomarán potencialmente una apariencia significativamente diferente.

Nivel de método. Una realización inserta la instrucción o instrucciones de lanzamiento de excepción en el nivel de método (por ejemplo, método, procedimiento, función, clase, método de clase, objeto, estructura de datos, interfaz, etc.) 502. En ese caso cada vez que se llama (ejecuta, instancia, inicializa, etc.) al método desde algún lugar en el programa, la instrucción de lanzamiento de excepción se ejecuta, y no se ejecuta parte del método. En otros casos, el traductor puede determinar que incluso aunque el método (etc.) en general es inseguro, una invocación particular del método es segura. En un caso de este tipo, el traductor puede crear una instancia segura del método en el flujo de salida siempre que se llame la invocación segura, mientras se coloca una instrucción de lanzamiento de excepción en el método real.

Nivel de bloque básico. En otra realización, se coloca una instrucción o instrucciones de lanzamiento de excepción en el nivel 503 de bloque básico. En general, un bloque básico es una serie secuencial de instrucciones que una vez introducidas, se completarán secuencialmente. Insertar (o colocar) una instrucción de lanzamiento de excepción como la primera instrucción en un bloque básico que contiene código irresoluble asegura que cualquier manipulación de datos que tuviera lugar después de entrar en el bloque básico pero antes de encontrar el código irresoluble, no se ejecutará. Esto ayuda a asegurar que cada dato se modifica únicamente en unidades atómicas de bloques básicos. En una realización, como se muestra en la Figura 6, el nivel de bloque básico se identifica como sigue:

A. Ir a través del código línea a línea:

i. Cualquier instrucción que provoca que se ramifique lejos de un recorrido lineal de código 601-02 marca el final de un bloque básico, y la siguiente instrucción es el comienzo de un nuevo bloque 603-04 básico; más
ii. Cualquier lugar que reciba la ramificación lejos de un recorrido lineal de código 605-06 es el comienzo de un bloque básico.

B. Colocar una instrucción o instrucciones de lanzamiento de excepción delante de la primera instrucción en un bloque básico que contiene código irresoluble 607.
C. Además, como se muestra en la Figura 7, si una sección de código es tan defectuosa que el punto donde comienza una siguiente instrucción no es identificable (sección muerta) 701, entonces colocar la instrucción de lanzamiento de excepción delante del último bloque básico conocido por encima de la sección 702 muerta.

i. Pero si alguna otra sección se ramifica en la sección 703 muerta, entonces comprobar para ver si la instrucción donde conduce la ramificación es determinable, y si lo es iniciar un nuevo bloque básico allí 703; entonces continuar la línea mediante búsqueda de recorrido de línea para ramificaciones como se analiza en el párrafo A.

D. Si no se encuentran bloques básicos definibles en un método como se muestra en la Figura 8, entonces colocar la instrucción de lanzamiento de excepción en lugar del método 801.

Usando esta realización para determinar bloques básicos, la instrucción de lanzamiento de excepción en el método 500 anterior podría insertarse inmediatamente delante de la siguiente línea: “y := (200000*0,16)” 504.

Nivel de instrucción. En otra realización, la instrucción o instrucciones de lanzamiento de excepción reemplazan la instrucción o instrucciones reales de código irresoluble 501. Una realización de este tipo colocaría la instrucción de lanzamiento de excepción en el flujo de salida donde hubiera aparecido el código irresoluble traducido si se hubiera traducido satisfactoriamente.

Además, la inserción y reemplazo de código irresoluble con una instrucción o instrucciones de lanzamiento de excepción puede tener el mismo significado. Por ejemplo, si se inserta una instrucción de lanzamiento de excepción como la primera línea en un bloque básico, es posible (dependiendo del manejo de excepción) que no se ejecutará ninguna otra línea en el bloque básico. En una realización de este tipo, no importaría si la instrucción de lanzamiento de excepción se insertó delante de la primera línea de código en el bloque básico, reemplazó la primera línea de código en el bloque básico, o reemplazó el bloque básico enero. En los tres casos, las instrucciones posteriores en el bloque básico no se ejecutarían (de nuevo, esto supone que ningún manejador devuelve la ejecución del saldo del bloque básico, que se analiza a continuación).

Finalmente, en algunos casos, la instrucción o instrucciones de lanzamiento de excepción y/o manejo se insertan en el flujo de salida en un nuevo bloque básico con el flujo de control y/o flujo de datos que encontrarían de otra manera unos errores de traducción irresolubles, dirigidos al nuevo bloque básico. Pueden emplearse también otros métodos para dirigir la ruta ejecución a las instrucciones insertadas. El punto es dirigir la ruta de control a las instrucciones insertadas en el caso de errores de traducción irresolubles.

Módulo o módulos de inserción de instrucción de lanzamiento de excepción

Para fines de ilustración, y no para limitar el alcance de la invención, se analiza un ejemplo de un traductor que contiene múltiples sub-unidades de traducción. La funcionalidad analizada y las combinaciones de funcionalidad analizadas no necesitan estar presentes en ninguna realización particular.

En una realización, tanto ambos módulos de inserción de instrucción o instrucciones de lanzamiento de excepción de nivel de método como de nivel de bloque básico se indican para un traductor de código de lenguaje intermedio. Como se muestra en la Figura 9, el flujo de entrada 900, se traduce en un flujo de salida 907, en una serie de cinco sub-unidades 902-906 de traducción. (Un proceso de traducción es una manifestación agregada de una o más sub-unidades de traducción. Véase anteriormente “Errores de Traducción Irresolubles”). Durante la traducción, si se encuentra código irresoluble, una sub-unidad de traducción de identificación llama a un módulo (sub-unidad) de inserción de instrucción de lanzamiento de excepción para generar una instrucción de lanzamiento de excepción en una posición 908-909 indicada. Deberá entenderse que estos módulos 908-909 de inserción pueden lanzar y/o manejar instrucciones de excepción insertadas en el flujo de salida.

En la primera sub-unidad 902 de traducción, el flujo de entrada se explora para determinar los tipos de parámetros y locales variables. Tras identificar el código irresoluble, la primera sub-unidad de traducción llama al módulo 909 de inserción de instrucción o instrucciones de lanzamiento de excepción de nivel de método, que indica la localización del método que contiene código irresoluble. En esta realización, la sub-unidad determina que la ejecución de alguna parte del método sería inaceptable puesto que hay código irresoluble que tiene un efecto global en el método. (En otra realización, esta determinación podría realizarse en un módulo de inserción de instrucción o instrucciones de lanzamiento de excepción en respuesta a la notificación de código irresoluble.) El módulo 909 de inserción de instrucción o instrucciones de lanzamiento de excepción de nivel de método genera una instrucción e inserta la instrucción o instrucciones en una localización estratégica en el flujo de salida, de modo que se ejecuta en el caso de que se llame al método afectado desde algún lugar en el código traducido. Esto puede conseguirse colocando la instrucción o instrucciones de lanzamiento de excepción en una única localización en el flujo de salida y teniendo todas las llamadas del método afectado dirigidas a la única localización. Puede conseguirse también colocando la instrucción o instrucciones de lanzamiento de excepción en cada localización donde se llama al método. En algunos casos, una instrucción de lanzamiento de excepción de nivel de método puede determinar que incluso aunque un método sea inseguro en general, una invocación particular de ese método es segura. En un caso de este tipo, el traductor puede generar una instancia segura de la invocación del método segura en la localización donde se llama a la invocación del método segura (este proceso se denomina un método “interior”) mientras coloca de otra manera una instrucción o instrucciones de lanzamiento de excepción en el método real. Puede haber muchas otras maneras conocidas en la técnica para dirigir una ruta de ejecución.

En la segunda sub-unidad 903 de traducción, si la segunda sub-unidad de traducción no puede formar bloques básicos apropiados para un método dado, entonces la segunda sub-unidad de traducción no puede determinar ninguna porción segura del método que pueda ejecutarse. (En otra realización, se determinan bloques básicos en procedimientos, objetos, funciones, código spaghetti, o etc.) Aunque los valores del método globales son seguros (como se describe en la primera subunidad traducción), la segunda sub-unidad 903 de traducción no puede segregar el código irresoluble en un bloque básico determinable, que deja todo el método en riesgo. La segunda sub-unidad de traducción invoca el módulo 909 de inserción de instrucción o instrucciones de lanzamiento de excepción de nivel de método para generar instrucción o instrucciones de lanzamiento de excepción en una

localización estratégica de modo que la instrucción o instrucciones de lanzamiento de excepción se ejecutan en el caso de que se llame al método.

En la tercera sub-unidad 904 de traducción, puesto que la traducción ha transcurrido la primera y la segunda sub-unidades de traducción, los bloques básicos son determinables. Esto permite que se segregue el código irresoluble en un único bloque básico y permite que se utilice el método en casos generales. En esta realización, la tercera sub-unidad de traducción traduce cada bloque básico convirtiendo el código de lenguaje intermedio en una representación de árbol de sintaxis abstracto, y tras identificar el código irresoluble, emplea el módulo 908 de inserción de instrucción o instrucciones de lanzamiento de excepción de nivel de bloque básico. La tercera sub-unidad de traducción, determina que la ejecución de cualquier parte de un bloque básico que contiene código irresoluble es inaceptable. (En otra realización, esta determinación podría realizarse en un módulo de inserción de instrucción de lanzamiento de excepción). El módulo 908 de inserción de instrucción o instrucciones de lanzamiento de excepción de bloque básico genera una instrucción e inserta la instrucción o instrucciones a medida que se ejecuta la primera instrucción cuando se ejecuta ese bloque básico. En una realización, la instrucción o instrucciones de lanzamiento de excepción se insertan en lugar de la primera instrucción en el bloque básico. En otra realización, la inserción se realiza en lugar de y en sustitución de todo el bloque básico (esta realización supone que ningún manejador permitirá la recuperación de alguna subparte del bloque básico afectado).

En otra realización, se emplea un módulo de instrucción o instrucciones de lanzamiento de excepción de nivel de instrucción para insertar instrucciones en lugar del código irresoluble encontrado en el nivel de instrucción irresoluble individual.

En otra realización, se emplea un único módulo de inserción de instrucción o instrucciones de lanzamiento de excepción para insertar las instrucciones de lanzamiento de excepción en código traducido en todos los niveles que comprenden la inserción de nivel de instrucción, la inserción del nivel de método y la inserción del nivel de bloque básico. Un módulo de inserción de instrucción o instrucciones de lanzamiento de excepción deberá llamarse también una sub-unidad de traducción de inserción de instrucción o instrucciones de lanzamiento de excepción (o sub-unidad de inserción), y una instrucción de excepción insertada puede ser también una instancia de objeto de lanzamiento de excepción de una clase de lanzamiento de excepción.

Finalmente, los módulos 908-909 de inserción de instrucción de lanzamiento de excepción ilustrados, pueden insertar también nuevos bloques (métodos, instrucciones, o etc.) básicos en algún lugar en el flujo de salida, conteniendo tales nuevos bloques básicos instrucción o instrucciones de lanzamiento de excepción y/o manejo. En una realización de este tipo, el flujo de control en el flujo de salida se adaptará para dirigir la ejecución a tal nuevo bloque básico en el caso de que la ruta de ejecución encontrara de otra manera un error de traducción irresoluble, para un caso de ruta de ejecución dada. Se analiza a continuación un ejemplo más detallado de esta realización en “Verificación de Código”. Sin embargo, esta realización se aplica no únicamente a verificación de código, sino a todas las sub-unidades de traducción. Véase la Figura 3.

Lanzamiento de excepciones de manejo

Una realización inserta una instrucción de lanzamiento de excepción convencional para todos los casos cuando se encuentra código irresoluble en el flujo de entrada. En una realización de este tipo el código traducido que contiene la instrucción de lanzamiento de excepción convencional lanza la excepción tras la ejecución de la instrucción. En una realización de este tipo la excepción lanzada provoca que el programa traducido en ejecución detenga la ejecución y devuelva el control al sistema operativo.

Sin embargo, no es inusual en la técnica anterior que un programa tenga una o más instrucciones de captura de excepción existentes en el código traducido. Estas instrucciones de tipo intento de captura, y otras formas de instrucciones de manejo pueden encontrarse en muchos lenguajes modernos tales como ADA, C++ y Java. Véase H.M. Deitel, C++ How To Program, Prentice-Hall, Inc. 1994, y véase Gary Cornell, Core Java, segunda edición, SunSoft Press, 1997. Estas instrucciones de captura de excepción existen a menudo a través de todos los múltiples niveles (por ejemplo, llamadas de método anidadas) en la estructura de programa y se analizan en la técnica anterior. Además, estas rutinas de captura de excepción de la técnica anterior no se escribieron específicamente para adaptar una instrucción de lanzamiento de excepción ejecutada insertada en código traducido en respuesta a identificar código irresoluble en un flujo de entrada. Posteriormente, cuando se ejecuta y se lanza una instrucción de lanzamiento de excepción que se inserta mediante la realización ilustrada, la excepción lanzada se propagará de vuelta a través de los múltiples niveles de la estructura de programa, identificando potencialmente instrucciones (y/o manejadores) de captura de excepción de la técnica anterior en la ruta de vaciado de pila, sin encontrar una captura o manejador a medida.

En otra realización, además de identificar código irresoluble en un flujo de entrada, se inserta tanto una instrucción o instrucciones de lanzamiento de excepción, como una instrucción o instrucciones de captura de excepción en el código traducido. En una realización de este tipo, la instrucción de captura de excepción contiene o indica un manejador apropiado para la instrucción de lanzamiento de excepción insertada correspondiente. Un manejador contenido o indicado de este tipo proporciona una salida elegante específica de contexto del programa o una recuperación de excepción específica de contexto.

En otra realización, la instrucción de lanzamiento de excepción insertada se lanza tras la ejecución y posteriormente se captura mediante el manejador de captura de excepción insertado existente en el programa traducido, o si el código traducido no contiene manejadores de captura de excepción, la excepción se propaga todo el camino al sistema operativo para cualquier manejo potencial.

En otra realización, como se muestra en la Figura 10, se coloca una instrucción o instrucciones (y/o manejador) de captura de excepción en la ruta de ejecución para el método, bloque básico o sección de código traducido que contiene la instrucción de lanzamiento de excepción insertada. En una realización de este tipo, las instrucciones (y/o manejador) 1001 de captura de excepción se colocan en el flujo de salida además de colocar la instrucción 1002 de lanzamiento de excepción en el flujo de salida. La rutina (o manejador) 1003 de captura devuelve el control a cualquiera de (i) el siguiente bloque 1004 básico en la ruta de ejecución, (ii) la siguiente línea en el principal después de la llamada al método que contiene la instrucción, o (iii) una rutina de manejo de excepción especializada que facilita la recuperación de excepción controlada en respuesta a instrucciones de lanzamiento de excepción.

Otra realización coloca un manejador de captura en el flujo de salida además de colocar la instrucción de lanzamiento de excepción en el flujo de salida. En una realización de este tipo, como se muestra en la Figura 11, un manejador de captura de ejecución notifica al usuario acerca de la excepción ejecutando una interfaz de usuario gráfica (GUI) en ventana, donde la GUI incluye un análisis y divulgación acerca del método o clase en que se encontró 1100 la instrucción de lanzamiento de excepción. En una realización de este tipo, este análisis y divulgación de GUI se toma desde los comentarios de los desarrolladores de código fuente, que incluye comentarios acerca del método 1101, bloque básico 1102, o instrucción donde se encontró 505-506 el código irresoluble. En otra realización, tal análisis y divulgación se toman desde metadatos en el flujo de entrada. En una realización de este tipo, los metadatos dirigen el análisis y divulgación pertinentes a la instrucción, método, bloque básico, etc., que contenían el código irresoluble. Como se muestra en la Figura 5, el desarrollador de código fuente coloca apropiadamente comentarios y/o metadatos en el código fuente 505-506 que se traducen a través del flujo de entrada presente y se incorporan mediante una realización en el manejador de captura. (Véase a continuación “Motor de metadatos”).

En otra realización, un manejador de instrucción de captura de GUI insertado presenta al usuario con maneras alternativas para continuar. En una realización de este tipo, como se muestra en la Figura 11, el manejador de captura de GUI ofrece una lista de opciones 1103 al usuario --salir de programa, devolver control a la siguiente instrucción, devolver control al siguiente bloque básico en ruta de ejecución dudosa, devolver control al siguiente método en ruta de ejecución dudosa o devolver control a la siguiente línea en el principal después de la llamada al método que contiene la excepción.

En otra realización, como se muestra en la Figura 10, se inserta una captura de excepción y una instrucción o instrucciones 1001-02 de lanzamiento de excepción tras identificar código irresoluble, y las instrucciones insertadas se lanzan y capturan posteriormente durante la ejecución, y la instrucción o instrucciones de captura identifican la clase de la captura de excepción, y determinan una clase de manejador de excepción basándose en la clase de la captura 1003 de excepción. En una realización de este tipo, la clase de manejador para la clase de excepción lanzada se identifica desde la biblioteca de clases. Este enlace de clase de excepción puede hacerse en tiempo de traducción, o posteriormente en tiempo de ejecución. En otra realización de este tipo, la clase de manejador para la clase de excepción lanzada se obtiene dinámicamente desde una biblioteca de clase remota enlazando o intercambiando mensajes con un servidor local o remoto en tiempo de ejecución. En una realización de este tipo, se ejecuta la clase de manejador identificada, enlazada u obtenida. Dada una realización de este tipo, como se muestra en la Figura 5, un desarrollador de código fuente indica una clase 507 de instrucción de lanzamiento de excepción deseada para que se lance, incluyendo comentarios y/o metadatos apropiados en el código fuente. Por lo tanto, un desarrollador de código fuente conoce que una clase 507 de excepción indicada provocará que se emplee localmente un manejador dado, o que se obtenga un manejador dado remotamente para manejar una clase de excepción lanzada indicada 1002 de este tipo. Una determinación de manejador dinámico de este tipo, además de versatilidad de manejo, proporciona una metodología para obtener una versión compatible de entorno de ejecución anterior de un método (procedimiento, instrucción, etc.). La determinación de manejador de tiempo de ejecución de este tipo se analiza además a continuación.

Metadatos y seguridad

En una realización, un traductor modifica su comportamiento de traducción basándose en la fuente del primer flujo de entrada de lenguaje. En una realización de este tipo, el traductor determina que alguna porción del flujo de entrada es ambigua pero aún traducible. El código que es ambiguo pero aún traducible se denomina “código dudoso.” En otra realización, la sección de código ambiguo cae dentro de una de diversas clases de código dudoso ambiguo pero traducible. Tal código dudoso es código que se recibe en un formato que podría decirse que es esperado o traducible, pero por una razón u otra permanece dudoso o potencialmente inseguro. El código dudoso es un subconjunto del universo de código irresoluble.

En una realización de este tipo, si el flujo de entrada que contiene el código dudoso es desde una fuente fiable o segura, el traductor traduce el código dudoso en el segundo flujo de salida de lenguaje sin colocar una instrucción de lanzamiento de excepción en el flujo de salida. El código dudoso pero traducible se traduce al segundo lenguaje sin

que se inserte una instrucción de lanzamiento de excepción, puesto que el código dudoso es desde una fuente confiable.

Un ejemplo de una realización de este tipo, como se muestra en la Figura 12, es un identificador (o asignación a un identificador) en el (primer lenguaje) flujo de entrada que el traductor no puede comprobar 1202 el tipo con seguridad. Una realización de este tipo determina basándose en la fuente de desarrollador de software del flujo 1201 de entrada que el código dudoso de otra manera debería traducirse sin colocar una instrucción (o manejador) de lanzamiento de excepción en el flujo de salida.

Otro ejemplo de código dudoso de este tipo, es un flujo de entrada que contiene un enlace a un recurso 1203 remoto, siendo el recurso enlazado un recurso remoto donde los identificadores en el flujo de entrada están definidos o donde las bibliotecas de clases solicitadas mediante el flujo de entrada están disponibles. Si el enlace está sin funcionamiento en el tiempo de traducción estos identificadores no serían verificables. Sin embargo, puesto que el flujo de entrada es desde una fuente 1201 confiable, una realización ilustrada completa la traducción y crea código traducido como si el enlace estuviera en funcionamiento y los identificadores se hubieran identificado, y sin colocar una instrucción de lanzamiento de excepción en el flujo de salida. En una realización de este tipo, si fuera necesario, se coloca un manejador en el flujo de salida para facilitar el enlace en tiempo de ejecución en el caso de que se acceda a un enlace confiable mediante el código traducible de ejecución en tiempo de ejecución.

En una realización de este tipo, el traductor necesita una manera para identificar la fuente del flujo (o fichero) de entrada. En una realización, si el flujo de entrada se obtuvo a través de la red, la fuente del software se determina basándose en la identificación de la fuente 1204 de servidor de dominio (o un paquete de IP). En otra realización, la fuente del software se determina mediante un identificador (o metadato) contenido en el (primer lenguaje) flujo 1201, 1204, 1205, 1203 de entrada.

Otra realización determina que aunque un flujo (o fichero) de entrada no es desde una fuente confiable, en su lugar la fuente está ‘certificada’ mediante una fuente 1205 confiable. En una realización de este tipo, una certificación de fuente se publica mediante una fuente confiable. En otra realización, una biblioteca de certificación de fuente se actualiza periódicamente de manera dinámica usando metodologías similares a las actualizaciones de ficheros de definición de virus actuales. En otra realización, una certificación de fuente está disponible en un enlace (o servidor de mensajería) proporcionado mediante una fuente 1204 confiable, o una fuente confiable proporciona una clave a la fuente desconocida que cuando se recibe es la prueba de que la fuente desconocida está certificada 1205. Una clave de este tipo u otro identificador que proporciona la certificación se conoce con antelación o es descubrible mediante una realización a través de un enlace a un servidor 1204 remoto. Otra realización accede a un servidor de fuente confiable conocido o designado para determinar si un flujo de entrada es confiable o certificado.

En otra realización, los metadatos en el (primer lenguaje) flujo de entrada identifican su fuente 1201. En otra realización, los metadatos identifican el flujo de entrada como que está certificado mediante un número de fuentes que son todas fuentes 1205 confiables. En otra realización, la fuente de software se determina basándose en (i) la identificación de dominio de fuente, (ii) metadatos que contienen información de identificación, u (iii) otra certificación de fuente mediante cualquiera de métodos estáticos predeterminados (por ejemplo, tabla de fuentes confiables, o tabla de números de certificación confiables) o métodos de descubrimiento de tiempo de traducción (por ejemplo, abrir un enlace a la fuente confiable o intercambiar mensajes con una fuente confiable). Además, en otra realización tal información sensible se protege con medidas de seguridad como claves privadas-públicas u otras formas de encriptación o codificación. Para los fines de esta memoria descriptiva, la expresión fuente confiable significa una fuente que se determina mediante cualquier combinación de los métodos anteriormente descritos para que estén de acuerdo con alguna consideración positiva en cómo o si un flujo de entrada se traduce. Una fuente no confiable puede estar de acuerdo con consideración negativa en cómo se maneja, y una fuente desconocida puede tratarse sin consideración o tratarse como confiable o no confiable dependiendo de consideraciones de seguridad.

La identificación de fuente 1201 de software se usa como una manera para distinguir entre múltiples maneras potenciales para manejar código irresoluble o dudoso. El código 1202 irresoluble cae en clases de código irresoluble. Tales clases se dividen en las clases lógicas analizadas anteriormente (Errores de Traducción Irresolubles), y estas clases pueden distinguirse además mediante clases de código dudoso. Además, cómo se maneja una clase de código irresoluble (o dudoso) mediante una realización puede basarse en el nivel de confianza de la fuente. Por ejemplo, una clase de código irresoluble o dudoso se maneja basándose en el nivel de confianza asociado de la fuente. En una realización de este tipo, tras identificar código irresoluble o dudoso, si el software es desde una fuente 1201 confiable de nivel superior, la realización simplemente traduce el código dudoso sin limitación adicional, y traduce el código irresoluble en una instrucción o instrucciones de lanzamiento de excepción.

En una realización, si la traducción continúa o no tras identificar código irresoluble o dudoso, depende de dónde se obtiene el flujo de entrada o quién desarrolló el flujo de entrada. En una realización de este tipo, para casos de una fuente de software desconocida o no confiable, la traducción se detiene tras identificar código irresoluble o dudoso. En una segunda realización, la traducción continúa en el caso de una fuente desconocida o no certificada, y se inserta una instrucción de lanzamiento de excepción en lugar del código irresoluble o dudoso. En una tercera realización, en el caso de una fuente conocida o confiable, la traducción continúa tras identificar código dudoso o irresoluble, creando de esta manera código traducido que contiene instrucciones de lanzamiento de excepción en

lugar de código irresoluble y código traducido en lugar de código dudoso.

En otra realización, los metadatos (u otros identificadores) en el flujo de entrada sugieren o enumeran niveles de colocación para instrucciones 1206 de lanzamiento de excepción (método, bloque básico o instrucción). En una realización de este tipo, los metadatos se consolidan en un área del flujo 1207 de entrada (por ejemplo, el comienzo o el final) y describen el manejo de excepción para todo el programa. En una realización de este tipo, la inserción de instrucción o instrucciones de lanzamiento de excepción puede intentarse en primer lugar en todos los casos en el nivel de bloque básico (por defecto) 1206 y a continuación si los bloques básicos no son determinables, intentar entonces una inserción 1206 de nivel de método. En una realización de este tipo, los metadatos pueden enumerar condiciones especiales para inserción, como inserciones de nivel de instrucción para el enlazamiento 1210 de manejador de tiempo de ejecución. Una realización de este tipo podría poner por defecto bloques básicos en todos los casos excepto el enlazamiento en tiempo de ejecución para obtener manejadores de excepción localmente 1209 o remotamente 1210.

En otra realización de este tipo, los metadatos (u otros identificadores) sugieren o enumeran la instrucción o instrucciones de lanzamiento de excepción para cada método, objeto, procedimiento o instrucción de este tipo en el flujo de entrada. En una realización de este tipo, como se muestra en la Figura 5, los metadatos (u otros identificadores) se dispersan a través de todo el fichero o flujo de entrada, estando próximos y/o identificado cada uno tal método o bloque básico, y sugiriendo o enumerando una instrucción o instrucciones de lanzamiento de excepción o manejador para tal método 512 individual o bloque 507-511 básico en caso de errores de traducción. En una realización de este tipo, el desarrollador de código fuente documenta apropiadamente el código fuente con los metadatos o comentarios requeridos.

En otra realización, los metadatos (u otros identificadores) enumeran clases de manejadores para excepciones lanzadas en lugar de las áreas específicas del programa 507-512 (métodos, bloque básico, niveles de instrucción) en el caso de problemas de traducción. En una realización de este tipo, las clases o descripción de manejadores puede ser una identificación de una clase de manejador conocida estática o una clase de manejador descubrible dinámicamente (local o remotamente). Estos manejadores pueden obtenerse y colocarse en el flujo de salida en tiempo de traducción, o pueden obtenerse y ejecutarse en tiempo de ejecución dinámicamente cuando se lanza una excepción insertada.

En otra realización, la identificación del manejador de excepción enumerado en los metadatos para un caso 507-512 dado (método, instrucción o nivel de bloque básico) se accede (local o remotamente) en tiempo de ejecución 1208-1209 cuando se lanza la excepción. En una realización de este tipo, después de que se lanza 1409 una excepción y se captura mediante un manejador insertado en el código 1402-1403 traducido, el método de manejador abre dinámicamente un enlace (o envía un mensaje) que identifica la excepción lanzada basándose en un identificador de clase de excepción, y el enlace dinámico (o mensaje devuelto) devuelve una rutina de manejador para la excepción identificada. Este enlace en respuesta a una instrucción de lanzamiento de excepción ejecutada se distingue desde un manejador obtenido en tiempo de traducción.

En una realización de este tipo, el desarrollador de código fuente asocia una clase (o tipo) de excepción para cada área (método, procedimiento, instrucción, objeto, estructura de datos) con un enlace o ruta designados para acceder al manejador designado, y la clase de excepción asociada y enlace/ruta se colocan en el flujo de salida para uso mediante el entorno de ejecución. (Véase a continuación “Estructura de Datos de Manejo de Excepción”, para una realización de ejemplo). Esto reduce el volumen de comentarios/metadatos en el flujo de entrada puesto que se omiten los manejadores. Además, proporciona notificación al desarrollador de código fuente cuando se entra en contacto con un enlace para obtener un manejador. El desarrollador de código fuente puede usar esta notificación para mejorar el código fuente, comentarios, metadatos y/o código de lenguaje intermedio enviado a través de la red. Tal notificación al desarrollador de código fuente proporciona un bucle de realimentación para mejoras de programa.

Métodos de manejador de excepción

En una realización ilustrada mostrada en la Figura 13, se coloca un manejador 1301 de excepción en el segundo fichero de salida de lenguaje en lugar de la instrucción de lanzamiento de excepción analizada anterior. En lugar de insertar una instrucción o instrucciones de lanzamiento de excepción en el código traducido, se inserta una instrucción o instrucciones de manejador en el código traducido. En una realización de este tipo, el manejador se inserta en el nivel de instrucción, el nivel de método o el nivel de bloque básico usando los mismos métodos que se han descrito anteriormente para insertar las instrucciones de lanzamiento de excepción. En otra realización, el manejador de excepción se inserta en lugar del código irresoluble en algún lugar en la ruta de ejecución que permite una salida elegante del programa, o permite el manejo de recuperación elegante como se ha descrito anteriormente para las instrucciones de captura de excepción, manejadores de captura o manejadores de excepción. Esta realización produce los mismos resultados funcionales que la realización de lanzamiento-captura-manejo anteriormente descrita (véase la Figura 10), sin las instrucciones de captura de lanzamiento. Por lo tanto, todas las otras realizaciones pueden importarse trivialmente en esta realización de instrucción de manejador (Figura 13).

En otra realización, un manejador 1301 de excepción insertado envía un mensaje a un dispositivo de salida que identifica el área del programa donde se localizó la entrada irresoluble que requiere el manejador de excepción que

describe potencialmente la función del área identificada del programa 1100.

En otra realización, la instrucción de manejador es un manejador convencional (por ejemplo, x86), o un manejador específico creado en respuesta a la entrada irresoluble que provoca la inserción del manejador.

En una realización que implementa un manejador de excepción específico en respuesta a entrada irresoluble, los metadatos en el flujo de entrada identifican clases potenciales de manejadores de excepción para áreas correspondientes del flujo de entrada. En una realización de este tipo, la biblioteca de clases para clases potenciales de manejadores de excepción está localizada en local o en remoto (como se ha descrito anteriormente), y puede enlazarse en tiempo de traducción u obtenerse en tiempo de ejecución tras ejecutar la instrucción o instrucciones de manejador.

Finalmente, la expresión instrucción o instrucciones de manejo incluirá una instrucción de manejo en solitario 1300, o una combinación 1001, 1003 de instrucción o instrucciones de captura-manejo.

Motor de metadatos

En párrafos anteriores en esta memoria descriptiva incluyendo “Metadatos y seguridad”, varias realizaciones usaron metadatos, identificadores y comentarios de los desarrolladores de código fuente en el flujo de entrada para ayudar a implementar un número de características de las realizaciones ilustradas. Además, una realización anteriormente analizada mencionó que la instrucción o instrucciones de lanzamiento de excepción pueden colocarse en el segundo flujo de salida de lenguaje en lugar del código irresoluble si ese segundo flujo de salida de lenguaje es (i) el flujo de salida de la primera traducción en una serie de N traducciones, (ii) la última traducción en una serie de N traducciones, o (iii) cualquier traducción en la serie de traducciones de una a N, donde N es mayor que o igual a uno.

Para que los metadatos, identificadores y comentarios de los desarrolladores de código fuente se propaguen a través de N niveles de traducción, cada traductor en las N fases de traducción debe propagar esta información en un formato reconocible en un flujo de salida de código de traducción.

Por ejemplo, en el caso de un proceso de traducción de dos fases (por ejemplo, fase 1: código fuente a código de lenguaje intermedio; fase dos: código de lenguaje intermedio a código objeto), los metadatos, identificadores y/o comentarios de los desarrolladores de código fuente deben pasar desde el código fuente en el código de lenguaje intermedio donde pueden usarse mediante una realización para crear una o más de las características descritas (por ejemplo, 1100-1103, 1409, 1402-1403, 1406, 1001-1003, 1201, 1203-1209, 1301) en el código objeto. Aunque el nivel deseado de especificidad debe pasar a través de sucesivas fases de traducción, puede emplearse la codificación y compresión para reducir el volumen del flujo de entrada y del flujo de salida.

En una realización, como se muestra en la Figura 9, un traductor incluye una sub-unidad para colocar los metadatos, identificadores y/o comentarios de los desarrolladores de código fuente en el flujo 910 de salida junto con el código traducible. Por simplicidad, la sub-unidad que coloca los metadatos, identificadores y/o comentarios de los desarrolladores de código fuente en el flujo de salida se denomina una sub-unidad de metadatos.

Finalmente, los metadatos y/o comentarios contenidos en el código fuente, flujo de entrada y/o el flujo de salida de las fases posteriores de traducción deberán conocerse también como “indicación o indicaciones textuales declarativas”.

Módulos de inserción de instrucción o instrucciones de captura y manejador

En una realización, que inserta instrucción o instrucciones de captura de excepción y/o instrucciones de manejo de excepción en código traducido en respuesta a insertar la instrucción o instrucciones de lanzamiento de excepción en código traducido, se emplea un módulo de inserción de instrucción. En una realización de este tipo, como se ilustra en la Figura 9, el mismo módulo (o sub-unidad) que inserta la instrucción o instrucciones de lanzamiento de excepción en el código 908-909 traducido, inserta la instrucción de captura de excepción y/o la instrucción o instrucciones de manejo de excepción en el código traducido.

En otra realización, se usa una sub-unidad separada para insertar cada tipo de código insertado. Un módulo (o sub-unidad) de inserción de este tipo inserta instrucciones de lanzamiento de excepción en el código traducido, una segunda sub-unidad de este tipo inserta instrucciones de captura de excepción en el código traducido, y una tercera sub-unidad de este tipo inserta instrucciones de manejo de excepción en el código traducido.

En otra realización, una única sub-unidad inserta toda la instrucción o instrucciones en respuesta a código irresoluble, ya sean excepciones, capturas o manejadores e independientemente del nivel de inserción (instrucción, nivel, nivel de bloque básico o nivel de método).

Manejo de excepción y/o estructuras de datos

En una realización, como se muestra en la Figura 14, cada método (procedimiento, bloque básico, función, programa principal, etc.) en el flujo de salida mantiene una estructura de datos posiblemente vacía de información

1401 de manejo de excepción. En una realización de este tipo, esta estructura de datos de manejador de excepción enumera los bloques básicos protegidos en un método 1402-1403. Un bloque básico está protegido si contiene una instrucción (y/o manejador) de lanzamiento de excepción insertado en respuesta a identificar un error de traducción irresoluble en el flujo 1404-1405 de entrada. Un método se protege si contiene una instrucción (y/o manejador) de lanzamiento de excepción insertado en respuesta a identificar código irresoluble en el flujo de entrada. Un área de un programa (o sección, procedimiento, función, instrucción, objeto, biblioteca enlazada, etc.) está protegido si contiene una instrucción de lanzamiento de excepción insertada en respuesta a encontrar código irresoluble en el flujo de entrada. La estructura de datos de manejador enumera un filtro 1402, y un manejador 1406 para cada uno del bloque o bloques básicos protegidos enumerados de un método 1405. El filtro coincide una clase 1407 de excepción con una designación 1408 de manejador, y la designación de manejador identifica un manejador 1406 para invocar en el caso de que el filtro 1403 coincida con la clase 1409 de excepción de una excepción lanzada.

En respuesta a un lanzamiento de excepción desde un bloque 1409 protegido, el control pasa a la estructura 1401 de datos del manejador en el método donde se lanzó la excepción 1400. Si el filtro determina que el lanzamiento de clase de excepción coincide con un filtro, se ejecuta el manejador designado mediante el filtro.

Si no se encuentra coincidencia en el método 1400 actual, la estructura de datos del manejador de excepción en el método que llama al método actual se busca para observar si contiene un filtro que designe un manejador para la clase de excepción lanzada. La búsqueda continúa volviendo a la secuencia de llamada hasta que se encuentre un manejador. Si no se encuentra coincidencia, la traza de la pila se vacía y el programa se aborta.

En otra realización, se inserta una única estructura de datos de manejador en el flujo de salida para un programa dado. En una realización de este tipo, la estructura de datos de manejador enumera una lista de identificadores de clase de excepción, que identifica cada clase de excepción insertada en el flujo de salida en lugar del código irresoluble, y una designación de manejador correspondiente para cada clase de excepción de este tipo.

En otra realización, se insertan las estructuras de datos de manejador en el flujo de salida en el nivel de bloque básico, o nivel de instrucciones, etc., en respuesta a colocar instrucciones de lanzamiento de excepción en el flujo de salida.

En otra realización, un módulo (o sub-unidad de inserción de estructura de datos de manejador, o sub-unidad de inserción) de inserción de estructura de datos de manejador inserta la estructura de datos de manejador en el flujo de salida en respuesta a la inserción de la instrucción de lanzamiento de excepción. En otra realización, una sub-unidad de inserción inserta tanto la instrucción de lanzamiento de excepción como la estructura de datos de manejador en el flujo de salida.

Tras identificar una designación de manejador, la rutina del manejador se identifica y puede ejecutarse. La rutina del manejador puede estar presente en el flujo 703 de salida, obtenida localmente 1410 en una ruta indicada, u obtenida dinámicamente a través de un servidor 1411 de enlace o de mensajería.

En otra realización, las instrucciones de manejador (y/o instrucciones de captura) se colocan en la ruta de ejecución de un lanzamiento de excepción desde un método protegido. En otra realización, las instrucciones de manejador (y/o instrucciones de captura) se colocan en la ruta de ejecución de un lanzamiento de excepción desde una instrucción protegida. En otra realización, las instrucciones de manejador (y/o instrucciones de captura) se colocan en la ruta de ejecución de un lanzamiento de excepción desde un programa protegido. Al igual que las estructuras de datos de manejador, este manejador (y/o instrucciones de captura) pueden obtenerse y/o enlazarse en tiempo de traducción, u obtenerse y/o enlazarse en tiempo de ejecución cuando se lanza una excepción.

Intento de re-traducción en tiempo de ejecución

En otra realización, tras encontrar una instrucción o instrucciones irresolubles en el flujo de entrada, el traductor inserta la instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo) en el flujo de salida. En una realización de este tipo, el traductor inserta también una copia no modificada del código irresoluble en el flujo de salida.

En una realización de este tipo, tras la ejecución posterior de las instrucciones de manejo en el flujo de salida, las instrucciones de manejo de ejecución solicitan un intento de re-traducción de la copia no modificada del código irresoluble en el flujo de salida. En una realización de este tipo, cuando el flujo de salida se ejecuta en tiempo de ejecución, las instrucciones insertadas solicitan que se realice un intento para traducir las instrucciones irresolubles. En algunos casos, esto puede resolver el error. Si el intento de re-traducción falla al resolver las instrucciones en cuestión, entonces se ejecuta la instrucción o instrucciones de lanzamiento de excepción (y/o instrucción o instrucciones de manejo). En una realización de este tipo, si fuera necesario, se coloca un manejador en el flujo de salida para facilitar el enlazamiento en tiempo de ejecución para facilitar la solicitud de re-traducción.

En una realización de este tipo, las instrucciones 409 de representación del segundo lenguaje insertadas, incluirían una copia del código irresoluble junto con las instrucciones que solicitan un intento 1003, 1301 de re-traducción.

Verificación de código

Esta memoria descriptiva se ha centrado principalmente en errores de traducción irresolubles en la forma de instrucciones irresolubles en el flujo de entrada. Sin embargo, las instrucciones de lanzamiento de excepción y/o de manejo pueden insertarse también en el flujo de salida tras encontrar errores de traducción irresolubles de muchos tipos. Véase anteriormente “Errores de Traducción Irresolubles.” Tales errores de traducción irresolubles no están limitados. Pueden ser cualquier condición en el flujo de entrada que un traductor determina que requiere instrucción o instrucciones de lanzamiento de excepción y/o de manejo en el flujo de salida.

Por ejemplo, la instrucción o instrucciones de lanzamiento de excepción y/o de manejo pueden insertarse para violaciones de reglas de verificación de código durante verificación de código. Los verificadores de código típicamente tienen un número reglas de verificación que deben verificarse para ayudar a asegurar que un programa es seguro para ejecución. Las reglas de verificador de código varían sustancialmente de distribuidor a distribuidor y de versión de distribuidor a versión de distribuidor de un compilador u otro sistema de traducción. En verificación de código, el flujo de entrada se comprueba para ver si viola o no alguna de estas reglas de verificación. Un tipo de regla de verificación de código, entre otros, que los verificadores de código hacen cumplir son las *reglas de restricción de tipo*. Durante la verificación de código, si se viola una de estas reglas de verificación de tipo, se genera un error de traducción irresoluble.

Un ejemplo para usar un verificador de código para comprar restricciones de tipo implica el uso de una pila. En general, la pila puede usarse mediante un programa como una pizarra intermedia para almacenar información de estado, argumentos e información de flujo de control. Los verificadores de código comprueban los tipos de argumento introducidos en una pila para determinar si coinciden con los tipos de argumento requeridos mediante la siguiente instrucción que extrae un argumento desde la pila. Las instrucciones introducen argumentos en y sacan argumentos de la pila. Los bloques básicos a menudo introducen argumentos en la pila y a continuación saltan a otro bloque básico. Las instrucciones de ramificación introducen la posibilidad de encontrar un bloque básico desde diferentes lugares en el programa. Como se ha analizado anteriormente (Identificar Áreas de Código Bloqueado mediante Instrucciones de Lanzamiento de Excepción), la ramificación a otras instrucciones a menudo crea el final de un bloque básico y el comienzo de otro. El verificador determina si las expectativas de un siguiente bloque básico son consistentes con el estado de la pila.

En un sentido general, el verificador de código atraviesa el flujo de entrada y determina si el estado de la pila puede usarse mediante cada bloque básico sucesor dadas las múltiples rutas de ejecución que pueden encontrarse en cualquier flujo de entrada dado. Si un bloque básico objetivo sucesor necesita un estado de pila diferente del producido mediante un bloque básico predecesor, entonces una instrucción en ese bloque básico no puede obtener un argumento requerido desde la pila. En ese caso el verificador encuentra un tipo que es ilegal para las necesidades del bloque básico (instrucción, método, etc.), e inserta una instrucción o instrucciones de lanzamiento de excepción y/o de manejo.

Un verificador sigue pasando siempre que sea posible, a través del flujo de entrada, para determinar un esquema en el que el estado de la pila sería válido para un flujo de entrada dado. Si en algún punto, el verificador no puede determinar un estado de pila para una ruta de ejecución que es válida, entonces se inserta una instrucción o instrucciones de lanzamiento de excepción y/o de manejo en el flujo de salida. Tras encontrar un error de este tipo descubierto mediante el verificador durante el flujo de control y/o análisis de flujo de datos, se inserta una instrucción o instrucciones de lanzamiento de excepción y/o de manejo en el flujo de salida para esa ruta de ejecución.

Durante la verificación de código se encuentran errores de traducción irresolubles como resultado de un error de flujo de control y/o de flujo de datos encontrado durante la verificación de tipo. Un error de flujo de control y/o de flujo de datos de este tipo puede tener lugar cuando una ruta de ejecución de programa alcanza las mismas instrucciones desde dos lugares diferentes. Por ejemplo, como se muestra en la Figura 15, puede determinarse que el flujo de entrada contiene un primer bloque 1501 básico y un segundo bloque 1502 básico, ambos de los cuales saltan al mismo bloque 1503 básico objetivo. En un caso de este tipo, un verificador 310 de código comprueba para observar si el estado de la pila 1506 antes de saltar desde el primer bloque básico al bloque básico objetivo es consistente con el estado de la pila antes de saltar desde el segundo bloque básico al bloque básico objetivo. Una regla de verificación de tipo tal, requiere que los tipos variables que se introdujeron en la pila antes de saltar al bloque básico objetivo desde el primer bloque básico, sean consistentes con los tipos de variables introducidas en la pila antes de saltar al bloque básico objetivo desde el segundo bloque básico.

Por ejemplo, una ruta puede introducir un entero y un tipo float en la pila y a continuación saltar a una localización objetivo. Otra ruta puede introducir un puntero y una variable de punto flotante en la pila y a continuación saltar a la misma localización objetivo. Puesto que estas dos rutas separadas saltan a la misma instrucción o instrucciones, y los tipos de variables en la pila son tipos diferentes, la regla de tipo se viola.

Para cualquier instrucción dada (bloque básico, método, etc.) la ruta de ejecución puede llegar allí linealmente, o puede saltar allí directamente. Si llegan diversas rutas de ejecución potenciales en una instrucción desde diferentes lugares en un programa, entonces estas diversas rutas de ejecución potenciales pueden haber introducido valores con tipos inconsistentes en la pila compartida. El verificador requiere que los elementos introducidos en la pila y requeridos mediante una instrucción objetivo (bloque básico, método, etc.) sean los mismos cuando llegan a esa instrucción objetivo. Suponiendo que un verificador de código ha determinado que dos estados de pila diferentes

vienen desde dos rutas diferentes que convergen en la misma instrucción objetivo (bloque básico, método, etc.), el verificador determina un fallo de regla. Esto es solo una manera de implementar un verificador. Un experto en la materia apreciaría que esto es solo una posible manera para implementar un verificador, y que cada implementación de este tipo podría tener muchos otros casos donde una regla de verificación se viola por código.

En la realización ilustrada, la primera ruta de ejecución se verifica y traduce en el flujo de salida, y la segunda ruta de ejecución da como resultado que se inserte una instrucción o instrucciones de lanzamiento de excepción y/o de manejo en el flujo 1504 de salida. Por lo tanto, el salto desde el primer bloque 1501 al bloque objetivo 1505 se traduce en el flujo de salida, y el salto desde el segundo bloque 1502 al bloque objetivo se traduce en un salto a una instrucción o instrucciones 1504 de lanzamiento de excepción y/o de manejo insertadas. Es interesante señalar que esto es un error de traducción irresoluble incluso aunque todas las instrucciones individuales sean directamente traducibles. En el siguiente ejemplo, el salto no siempre se dirige a las instrucciones insertadas.

En el siguiente ejemplo, un primer bloque 1501 básico finaliza en un salto condicional. El salto condicional, salta a dos localizaciones 1505, 1503 diferentes basándose en la condición. Durante la verificación, el primer salto al tercer bloque 1505 básico se determina para que sea verificable, pero el segundo salto a un bloque 1503 básico objetivo se determina para crear un error de traducción irresoluble basándose en un problema de flujo de control y/o de flujo de datos. En el segundo salto, tras encontrar el bloque básico objetivo, el verificador determina que los tipos de argumento/objeto en la pila son inconsistentes con los tipos de argumento/objetos requeridos mediante el bloque básico objetivo. Puesto que el estado de la pila desde el segundo salto es inconsistente con el estado de la pila requerido mediante el bloque básico objetivo, se inserta una instrucción o instrucciones de lanzamiento de excepción y/o de manejo en el flujo 1504 de salida. En un caso de este tipo, las instrucciones 1504 insertadas se ejecutan únicamente cuando el salto condicional es al bloque objetivo. El flujo de salida está dispuesto en consecuencia. Un bloque 1504 de fallo se está insertando puesto que el problema del flujo de control y/o del flujo de datos descubiertos durante la verificación que tiene lugar únicamente en el salto condicional desde el primer bloque básico al bloque básico objetivo.

En otro caso, el verificador intentará combinar el estado de la pila con los requisitos de estado de varios bloques básicos relacionados. Un caso de este tipo tiene lugar cuando los dos bloques básicos saltan al mismo bloque básico objetivo. El verificador intentará tratar de llegar a un estado de pila que es consistente para los bloques básicos que saltan al bloque básico objetivo y para el bloque básico objetivo. Esto se llama combinar la pila. Se identifica un problema de flujo de control y/o de flujo de datos en la verificación cuando los dos bloques 1501, 1502, básicos introducen diferentes tipos de argumentos en la pila, y a continuación saltan al mismo bloque 1503 básico objetivo.

Por ejemplo, suponiendo que uno de los bloques básicos introduce un puntero nulo en la pila, y que el bloque básico objetivo necesita un puntero a una clase foo. En una realización, esto es consistente, puesto que un puntero nulo nunca puede ser un puntero a una clase foo. Por lo que el puntero nulo no es inconsistente con el estado de la pila requerido por el bloque básico objetivo. Más tarde, otro bloque básico introduce una clase foo en la pila y a continuación salta al bloque básico objetivo. El verificador intentará tratar de llegar a un estado de pila que es consistente para los dos bloques básicos y para el bloque básico objetivo. En este caso, puesto que un foo es consistente con los tres bloques, la pila se combina. Por lo que se permite tanto al primer bloque 1501 básico como al segundo bloque 1502 básico saltar al bloque 1503 básico objetivo en el flujo de salida.

En otro caso, dos bloques básicos saltan al mismo bloque básico que hace una operación añadir. Como se muestra en la Figura 16, cada uno introduce diferentes operandos en la pila. El primer bloque 1601 básico inserta dos enteros en la pila antes de saltar al bloque 1603 básico que contiene la operación añadir. El segundo bloque 1602 básico inserta dos tipos float en la pila antes de saltar al bloque 1603 básico que contiene la operación añadir. Eso es ilegal. Viola una regla de la realización ilustrada, la de que los tipos que se introducen en la pila para un bloque básico objetivo son los mismos independientemente de la ruta que se tomó para llegar allí. El hecho de que se introduzca dos enteros antes de saltar desde una ruta, y se introduzca dos enteros antes de saltar desde la otra ruta, viola esta regla. Cualquiera en solitario está bien. Es aceptable añadir dos enteros, y es aceptable añadir dos tipos float, pero no es aceptable combinar estas dos rutas de ejecución dada esta regla.

En la realización ilustrada, se permitiría la primera ruta encontrada mediante el verificador (desde el primer bloque 1601 básico al bloque 1603 básico objetivo), y la segunda ruta de ejecución (desde el segundo bloque 1602 básico al bloque 1603 básico objetivo) se dirigiría a una instrucción o instrucciones 1604 de lanzamiento de excepción y/o de manejo en el flujo de salida. Este fallo se encontró en el flujo de control y/o en el flujo de datos a diferencia de cualquier instrucción ilícita individual. En la realización ilustrada, es el orden (primero en llegar primero en servir) que determina qué salto se permite y qué salto es para una instrucción o instrucciones de lanzamiento de excepción y/o de manejo recién insertada.

Además de colocar las instrucciones de lanzamiento de excepción y/o de manejo en lugar de (delante de, o dentro de) un bloque básico que contiene errores de traducción irresolubles, un traductor busca en el estado de la pila cuando atraviesa el flujo de entrada. En un caso de este tipo, una ruta de ejecución a un bloque básico dado puede no ser aceptable, pero otra puede permitirse. Por lo que incluso después de que se hayan determinado las instrucciones particulares en el flujo de entrada para que se resuelvan individualmente, el flujo de control de

programa, flujo de datos y/o la lógica pueden crear errores de traducción irresolubles adicionales.

El traductor trata de asegurarse que únicamente el código verificado termina en el flujo de salida. Esto puede conseguirse insertando la instrucción o instrucciones de lanzamiento de excepción y/o de manejo en (i) el bloque básico que salta a un bloque básico objetivo, (ii) en el bloque básico objetivo, o (iii) en un nuevo bloque básico insertado en el flujo de salida. Cualquiera de estos conseguirá el efecto--que asegura que se ejecute únicamente código verificado. Sin embargo, insertar un nuevo bloque básico que contiene la instrucción o instrucciones de lanzamiento de excepción y/o de manejo insertadas y se dirige el salto a esa localización parece rechazar en menor cantidad los programas. Deja tanto el bloque básico de origen como el bloque básico objetivo libres para interactuar con otros bloques básicos para otras rutas de ejecución verificables. Como se ha analizado anteriormente, el código confiado se inserta en el flujo de salida sin una instrucción o instrucciones de lanzamiento de excepción y/o de manejo insertadas.

Aunque un experto en la materia se daría cuenta de la importancia de todas las sub-unidades de traducción (Figura 3), es interesante observar verificación de código. Con este enfoque para verificación, se puede limitar la instrucción o instrucciones de lanzamiento de excepción y/o de manejo a rutas de ejecuciones que no pueden verificase. La instrucción o instrucciones de lanzamiento de excepción y/o de manejo son únicamente necesarias para rutas de ejecución específicas. La realización ilustrada, únicamente inserta la instrucción o instrucciones de lanzamiento de excepción y/o de manejo para rutas de código específicas, a diferencia de caer en grandes segmentos o todo el programa debido a que unas pocas rutas de flujo de control y/o de flujo de datos pueden no ser verificables.

Además, incluso cuando una instrucción o instrucciones de lanzamiento de excepción y/o de manejo se insertan en el flujo de salida, la instrucción o instrucciones de manejo y/o el entorno de tiempo de ejecución pueden intentar re-traducir la traducción fallida una vez más antes de abandonar. Véase anteriormente “Intento de Re-traducción en Tiempo de Ejecución” y véase anteriormente “Métodos de Manejador de Excepción”. Un verificador de código puede no poder resolver una situación durante la compilación. Si una regla no es entonces verificable, puede hacerse verificable más tarde cuando es necesaria para ejecución. En un caso de este tipo, puede estar disponible algo necesario desde un disco o de la red. Esto puede conseguirse mediante re-traducción y/o insertando la instrucción o instrucciones de manejo en el flujo de salida. Cuando el código es realmente necesario mediante un cliente, las condiciones pueden haber cambiado. Si un intento de manejador y/o re-traducción determina que ya no existe una condición de fallo, entonces esta forma de fallo vago y/o tardío puede proporcionar un éxito final. Esta situación es pertinente a la verificación.

Ocurre en el contexto de la verificación puesto que pueden hacerse disponibles nuevas clases y/o recursos o pueden de otra manera ser dependientes del tiempo/caso. Un flujo de entrada puede hacer referencia a algo que no está disponible cuando se compila el código (JIT o de otra manera), y normalmente que produciría una excepción de verificación. Pero con la realización ilustrada, a través de re-traducción y/o cuando se ejecuta un manejador insertado, aquellos recursos pueden a continuación estar disponibles.

Finalmente, será evidente para un experto en la materia en vista del análisis anterior, que en todos los casos de encontrar de errores de traducción irresolubles, (no solamente en verificación de código o instrucciones irresolubles), la instrucción o instrucciones de lanzamiento de excepción y/o de manejo insertadas pueden insertarse en un nuevo bloque básico en el flujo de salida con el flujo de control ajustado para dirigir errores de traducción irresolubles al mismo. Además, tales instrucciones insertadas pueden insertarse también delante de, o en lugar de cualquier error de traducción irresoluble como se analiza en esta memoria descriptiva (por ejemplo, “Identificar Áreas de Código Bloqueado por Instrucciones de Lanzamiento de Excepción”).

Entorno de operación ejemplar

La Figura 17 y el siguiente análisis se pretenden para proporcionar una descripción general breve de un entorno informático adecuado en el que puede implementarse la invención. Aunque la invención se describirá en el contexto general de instrucciones ejecutables por ordenador de un programa informático que se ejecuta en un ordenador y/o impresora informática, los expertos en la materia reconocerán que la invención puede implementarse también en combinación con otros módulos de programa. En general, los módulos de programa incluyen rutinas, programas, componentes, estructuras de datos, etc., que realizan tareas particulares o implementan tipos de datos abstractos particulares. Además, los expertos en la materia apreciarán que la invención puede ponerse en práctica con otras configuraciones de sistemas informáticos, incluyendo dispositivos portátiles, sistemas multiprocesador, electrónica de consumo basada en microprocesador o programable, miniordenadores, ordenadores centrales y similares. Las realizaciones ilustradas de la invención pueden ponerse en práctica también en entonos informáticos en red, o en ordenadores independientes.

Además, la realización ilustrada de la invención puede ponerse en práctica en cualquiera de todos los siguientes en solitario o en un entorno en red (inalámbrico o no): dispositivos informáticos portátiles, organizadores electrónicos, planificadores de día electrónicos, dispositivos electrónicos con pantallas, dispositivos conectados a pantallas, dispositivos conectados a impresoras, teléfonos celulares con exploradores en miniatura, buscapersonas textuales, dispositivos de inventario portátil, vehículos que contienen dispositivos en placa con visores, o dispositivos de cualquier tipo que representan texto o carácter para visualización o impresión.

Con referencia a la Figura 17, un sistema ejemplar para implementar la invención incluye un ordenador 1720 convencional (tal como un ordenadores personales, PC de bolsillo o de mano, decodificadores, servidores, ordenadores centrales y otra diversidad de ordenadores) incluye una unidad 1721 de procesamiento, una memoria 1722 de sistema, y un bus 1723 de sistema que acopla diversos componentes de sistema incluyendo la memoria de sistema a la unidad 1721 de procesamiento. La unidad de procesamiento puede ser cualquiera de diversos procesadores comercialmente disponibles, incluyendo Intel x86, Pentium y microprocesadores compatibles a partir de Intel y otros, incluyendo Cyrix, AMD y Nexgen; Alpha de Digital; MIPS de MIPS Technology, NEC, IDT, Siemens y otros; y el PowerPC de IBM y Motorola. Puede usarse también microprocesadores duales y otras arquitecturas multi-procesador como la unidad 1721 de procesamiento.

El bus de sistema puede ser cualquiera de diversos tipos de estructura de bus incluyendo un bus de memoria o controlador de memoria, un bus periférico, y un bus local que usa cualquiera de una diversidad de arquitecturas de bus convencionales tales como PCI, VESA, AGP, Microchannel, ISA y EISA, por nombrar unas pocas. La memoria de sistema incluye memoria 1724 de solo lectura (ROM) y memoria 1725 de acceso aleatorio (RAM). Un sistema de entrada/salida básico (BIOS), que contiene las rutinas básicas que ayudan a transferir la información entre elementos en el ordenador 1720, tal como durante el arranque, se almacena en la ROM 1724.

El ordenador 1720 incluye además una unidad 1727 de disco duro, una unidad 1728 de disco magnético, por ejemplo, para leer desde o escribir a un disco 1729 extraíble, y una unidad 1730 de disco óptico, por ejemplo, para leer un disco 1731 CD-ROM o para leer desde o escribir a otro medio óptico. La unidad 1727 de disco duro, la unidad 1728 de disco magnético, y la unidad 1730 de disco óptico están conectadas al bus 1723 de sistema mediante una interfaz 1732 de unidad de disco duro, una interfaz 1733 de unidad de disco magnético, y una interfaz 1734 de unidad de disco óptico, respectivamente. Las unidades y sus medios legibles por ordenador asociados proporcionan almacenamiento de datos no volátil, estructuras de datos, instrucciones ejecutables por ordenador, etc., para el ordenador 1720. Aunque la descripción de medio legible por ordenador anterior hace referencia a un disco duro, un disco magnético extraíble y un CD, debería apreciarse por los expertos en la materia que otros tipos de medios que son legibles mediante un ordenador, tales como cintas magnéticas, tarjetas de memoria flash, discos de vídeo digital, cartuchos Bernoulli y similares, pueden usarse también en el entorno operativo ejemplar.

Un número de módulos de programa puede almacenarse en las unidades y en la RAM 1725, incluyendo un sistema 1735 operativo, uno o más programas 1736 de aplicación, otros módulos 1737 de programa y datos 1738 de programa; además de una realización 1756.

Un usuario puede introducir comandos e información en el ordenador 1720 a través de un teclado 1740 y un dispositivo apuntador, tal como un ratón 1742. Otros dispositivos de entrada (no mostrados) pueden incluir un micrófono, joystick, controlador para juegos, antena parabólica, escáner o similares. Estos y otros dispositivos de entrada a menudo están conectados a la unidad 1721 de procesamiento a través de una interfaz 1746 de puerto serie que está acoplada al bus de sistema, pero puede estar conectada mediante otras interfaces, tal como un puerto paralelo, puerto de juegos o un bus serie universal (USB). Un monitor 1747 u otro tipo de dispositivo de visualización está conectado también al bus 1723 de sistema mediante una interfaz, tal como un adaptador 1748 de vídeo. Además del monitor, los ordenadores típicamente incluyen otros dispositivos de salida periféricos (no mostrados), tales como altavoces e impresoras.

El ordenador 1720 opera en un entorno en red usando conexiones locales a uno o más ordenadores remotos, tal como un ordenador 1749 remoto. El ordenador 1749 remoto puede ser un servidor, un encaminador, un dispositivo de pares u otro nodo de red común, y típicamente incluye muchos o todos los elementos descritos con relación al ordenador 1720, aunque únicamente se ha ilustrado un dispositivo 1750 de almacenamiento de memoria. Las conexiones lógicas representadas incluyen una red 1751 de área local (LAN) y una red 1752 de área extensa (WAN). Tales entornos de interconexión de red son comunes en oficinas, redes informáticas a nivel de empresa, intranets e internet.

Cuando se usa en un entorno de interconexión de red de LAN, el ordenador 1720 está conectado a la red 1751 local a través de una interfaz o adaptador 1753 de red. Cuando se usa en un entorno de interconexión de red WAN, el ordenador 1720 incluye típicamente un módem 1754 u otros medios para establecer comunicaciones (por ejemplo, mediante la LAN 1751 y una pasarela o servidor 1755 intermediario) a través de la red 1752 de área extensa, tal como internet. El módem 1754, que puede ser interno o externo, está conectado al bus 1723 de sistema mediante la interfaz 1746 de puerto serie. En un entorno en red, los módulos de programa representados con relación al ordenador 1720, o porciones de los mismos, pueden almacenarse en el dispositivo de almacenamiento de memoria remoto. Se apreciará que las conexiones de red mostradas son ejemplares y que pueden usarse otros medios para establecer un enlace de comunicaciones entre los ordenadores.

De acuerdo con las prácticas de los expertos en la materia de la programación informática, la presente invención se describe a continuación con referencia a actos y representaciones simbólicas de operaciones que se realizan mediante el ordenador 1720, a menos que se indique de otra manera. Tales actos y operaciones se denominan en ocasiones como que se ejecutan por ordenador. Se apreciará que los actos y operaciones representadas simbólicamente incluyen la manipulación mediante la unidad 1721 de procesamiento de señales eléctricas que representan bits de datos que producen una transformación resultante o reducción de la representación de la señal

eléctrica, y el mantenimiento de bits de datos en localizaciones de memoria en el sistema de memoria (incluyendo la memoria 1722 de sistema, disco 1727 duro, discos 1729 flexibles y CD-ROM 1731) para reconfigurar de esta manera o modificar de otra manera la operación del sistema informático, así como otro procesamiento de señales. Las localizaciones de memoria donde se mantienen los bits de datos son localizaciones físicas que tienen propiedades eléctricas, magnéticas u ópticas particulares que corresponden a los bis de datos. En el sistema informático ilustrado, se representa 1756 una realización ilustrada de la invención.

Habiendo descrito e ilustrado los principios de nuestra invención con referencia a una realización ilustrada, se reconocerá que la realización ilustrada puede modificarse en disposición y detalle sin alejarse de tales principios. Debería entenderse que los programas, procesos o métodos descritos en el presente documento no están relacionados o limitados a ningún tipo particular de aparato o software informático, a menos que se indique de otra manera. Diversos tipos de aparatos informáticos de fin general o especializados pueden usarse con o realizar operaciones de acuerdo con las enseñanzas descritas en el presente documento. Los elementos de la realización ilustrada mostrados en software pueden implementarse en hardware y viceversa.

Además, aunque se ilustra como implementada en un ordenador personal, la invención puede ponerse en práctica en otros aparatos informáticos digitales estén o no en red.

En vista de las muchas posibles realizaciones a las que pueden aplicarse los principios de nuestra invención, debería reconocerse que las realizaciones detalladas son ilustrativas únicamente y no deberían tomarse como que limitan el alcance de nuestra invención. En su lugar, reivindicamos como nuestra invención todas tales realizaciones como que puede entrar dentro del alcance de las siguientes reivindicaciones y equivalentes a las mismas.

REIVINDICACIONES

1. Un método para traducir un código de programa informático desde una primera representación de lenguaje a una segunda representación de lenguaje, comprendiendo el método:

traducir (400) instrucciones traducibles de un flujo de entrada en la primera representación de lenguaje en un flujo de salida a la segunda representación de lenguaje;
identificar (400) un código irresoluble en el flujo de entrada que un traductor no puede traducir desde la primera representación de lenguaje a la segunda representación de lenguaje;
y
colocar (400) al menos una segunda instrucción de representación de lenguaje en el flujo de salida en respuesta a identificar el código irresoluble en el flujo de entrada, en el que:

la al menos una segunda instrucción de representación de lenguaje colocada es una de una instrucción (1001, 1003) de manejo de excepción, un manejador (1301) de excepción o una instrucción (409, 1002) de lanzamiento de excepción; y
dicha colocación comprende colocar (409, 909, 908) la al menos una segunda instrucción de representación de lenguaje en una localización en el flujo de salida donde el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido el código irresoluble una instrucción traducible o donde un método que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si todo el método hubiera sido traducible o donde un bloque básico que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido traducible todo el bloque básico.

2. El método de la reivindicación 1 que comprende además:

lanzar la instrucción de lanzamiento de excepción colocada si se ejecuta el flujo de salida.

3. El método de la reivindicación 1, en el que dicha colocación comprende además:

dirigir la colocación de al menos una segunda instrucción de representación de lenguaje en el flujo de salida basándose en una indicación textual declarativa contenida en el flujo de entrada.

4. El método de la reivindicación 1, en el que dicha colocación comprende colocar tanto:

una instrucción (1002) de lanzamiento de excepción; como
una instrucción (1003) de manejo de excepción .

5. El método de la reivindicación 3, en el que la indicación textual declarativa designa que la al menos una segunda instrucción de representación de lenguaje reemplaza el código irresoluble.

6. El método de la reivindicación 3, en el que la indicación textual declarativa designa que la al menos una segunda instrucción de representación de lenguaje reemplaza un bloque básico que contiene el código irresoluble.

7. El método de la reivindicación 3, en el que la indicación textual declarativa designa que la al menos una segunda instrucción de representación de lenguaje reemplaza un método que contiene el código irresoluble.

8. El método de la reivindicación 1, que comprende además:

determinar desde una indicación textual declarativa en el flujo de entrada cuál al menos una segunda instrucción de representación de lenguaje colocar en el flujo de salida.

9. El método de la reivindicación 1, que comprende además:

obtener desde una biblioteca de al menos una segunda instrucción de representación de lenguaje disponible, la al menos una segunda instrucción de representación de lenguaje colocada en el flujo de salida.

10. El método de la reivindicación 1, que comprende además:

determinar un nivel para colocar la al menos una segunda instrucción de representación de lenguaje;
en el que la determinación de nivel para colocar la al menos una segunda instrucción de representación de lenguaje se realiza de entre un conjunto de niveles (1206) disponibles, incluyendo el conjunto de niveles disponibles al menos dos niveles distintos desde un grupo de niveles potenciales, comprendiendo el grupo de niveles potenciales un nivel de método, un nivel de instrucción, un nivel de bloque básico y un nivel de programa.

11. El método de la reivindicación 10, en el que una indicación textual declarativa indica el nivel para colocar la al menos una segunda instrucción de representación de lenguaje.

12. El método de la reivindicación 1, que comprende además:

colocar en el flujo de salida una copia no modificada del código irresoluble identificado.

13. El método de la reivindicación 12, que comprende además:

cuando se ejecuta el flujo de salida, ejecutar (1301) la al menos una segunda instrucción de representación de lenguaje colocada, invocando dicha ejecución un traductor en la copia no modificada del código irresoluble identificado desde el flujo de salida, provocando de esta manera que el traductor invocado intente traducir la copia no modificada del código irresoluble a una segunda instrucción de representación de lenguaje, y si la copia no modificada del código irresoluble identificado es entonces traducible a una segunda instrucción de representación de lenguaje, entonces se ejecuta la segunda instrucción de representación de lenguaje.

14. El método de la reivindicación 1 en el que dicha colocación comprende colocar la al menos una segunda instrucción de representación de lenguaje en un nuevo bloque básico en el flujo de salida.

15. El método de la reivindicación 3, en el que la indicación textual declarativa designa que la al menos una segunda instrucción de representación de lenguaje debería insertarse en un nuevo bloque básico.

16. Un medio legible por ordenador que almacena instrucciones operacionales para traducir un primer flujo de entrada de lenguaje en un segundo flujo de salida de lenguaje, comprendiendo las instrucciones:

instrucción o instrucciones para traducir (901) porciones traducibles del primer flujo de entrada de lenguaje en el segundo flujo de salida de lenguaje;
instrucción o instrucciones que identifican (901) un código irresoluble en el primer flujo de entrada de lenguaje que un traductor no puede traducir desde la primera representación de lenguaje a la segunda representación de lenguaje;
instrucción o instrucciones que colocan (901) al menos una segunda instrucción de representación de lenguaje en el segundo flujo de salida de lenguaje en respuesta a identificar el código irresoluble en el primer flujo de entrada de lenguaje, en el que la al menos una segunda instrucción de representación de lenguaje colocada es una de una instrucción de manejo de excepción, un manejador de excepción o una instrucción de lanzamiento de excepción; e
instrucción o instrucciones que colocan la al menos una segunda instrucción de representación de lenguaje en una localización en el flujo de salida donde el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido el código irresoluble una instrucción traducible o donde un método que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si todo el método hubiera sido traducible o donde un bloque básico que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido traducible todo el bloque básico.

17. El medio legible por ordenador de la reivindicación 16, en el que el primer flujo de entrada de lenguaje comprende además una indicación textual declarativa que indica dónde colocar en el flujo de salida una instrucción de lanzamiento de excepción.

18. El medio legible por ordenador de la reivindicación 16, en el que el primer flujo de entrada de lenguaje comprende además una indicación textual declarativa que indica dónde colocar en el flujo de salida el manejador de excepción.

19. El medio legible por ordenador de la reivindicación 16 que comprende además instrucciones para colocar una instrucción de manejo de excepción y una instrucción de lanzamiento de excepción en el segundo flujo de salida de lenguaje.

20. El medio legible por ordenador de la reivindicación 16, en el que una indicación textual declarativa indica colocar la al menos una segunda instrucción de representación de lenguaje en el flujo de salida en un método o un nivel de bloque básico.

21. El medio legible por ordenador de la reivindicación 16 que comprende además instrucciones para colocar en el segundo flujo de salida de lenguaje, una copia no modificada del código irresoluble identificado.

22. Un sistema informático que tiene un dispositivo de entrada de usuario, un primer flujo (401, 900) de entrada de lenguaje, un sistema operativo y un sistema (400) de traducción que opera bajo el control del sistema operativo, comprendiendo además el sistema informático:

un proceso (400) de traducción que opera bajo el control del sistema operativo, y que traduce el primer flujo de entrada de lenguaje en un segundo flujo (408, 907) de salida de lenguaje posterior a una entrada de usuario en el dispositivo de entrada de usuario;
una sub-unidad (400, 902-906) de traducción operacional para realizar alguna porción del proceso de traducción; y
una sub-unidad (400, 908-910) de traducción operacional para determinar que un código irresoluble encontrado en el primer flujo de entrada de lenguaje no puede traducirse en el segundo flujo de salida de lenguaje, y en respuesta a la determinación, insertar al menos una segunda instrucción de representación de lenguaje en el segundo flujo de salida de lenguaje, en el que la al menos una segunda instrucción de representación de lenguaje es una de una instrucción (1001, 1003) de manejo de excepción, un manejador (1301) de excepción o

una instrucción (1002) de lanzamiento de excepción, y en el que la al menos una segunda instrucción de representación de lenguaje se inserta en una localización en el flujo de salida donde el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido el código irresoluble una instrucción traducible o donde un método que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si todo el método hubiera sido traducible o donde un bloque básico que contiene el código irresoluble en el flujo de entrada se hubiera colocado en el flujo de salida si hubiera sido traducible todo el bloque básico.

23. El sistema informático de la reivindicación 22 en el que una indicación textual declarativa en el primer flujo de entrada de lenguaje indica dónde insertar la al menos una segunda instrucción de representación de lenguaje en el segundo flujo de salida de lenguaje.

24. El sistema informático de la reivindicación 22 en el que una indicación textual declarativa en el primer flujo de entrada de lenguaje indica insertar el manejador de excepción en un bloque básico en el segundo flujo de salida de lenguaje.

25. El sistema informático de la reivindicación 22 en el que una indicación textual declarativa en el primer flujo de entrada de lenguaje indica insertar el manejador de excepción en un nivel de método.

26. El sistema informático de la reivindicación 22 estando adaptado además para insertar en el flujo de salida una copia no modificada de la instrucción encontrada en el primer flujo de entrada de lenguaje que no puede traducirse en el segundo flujo de salida de lenguaje.

27. El sistema informático de la reivindicación 26 estando adaptado además para traducir la copia no modificada tras la ejecución del segundo flujo de salida de lenguaje.

Fig. 1, Técnica anterior

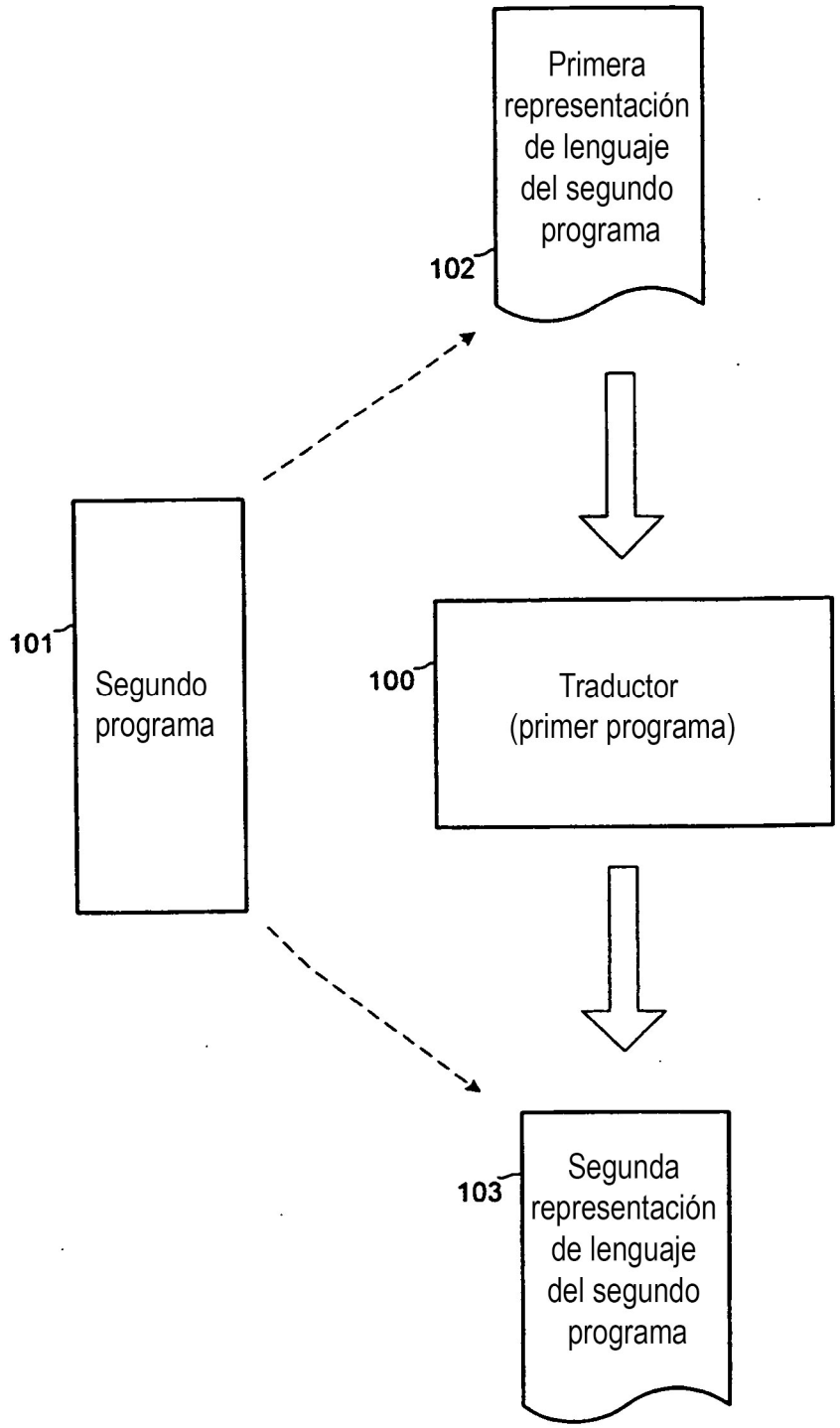


Fig. 2, Técnica anterior

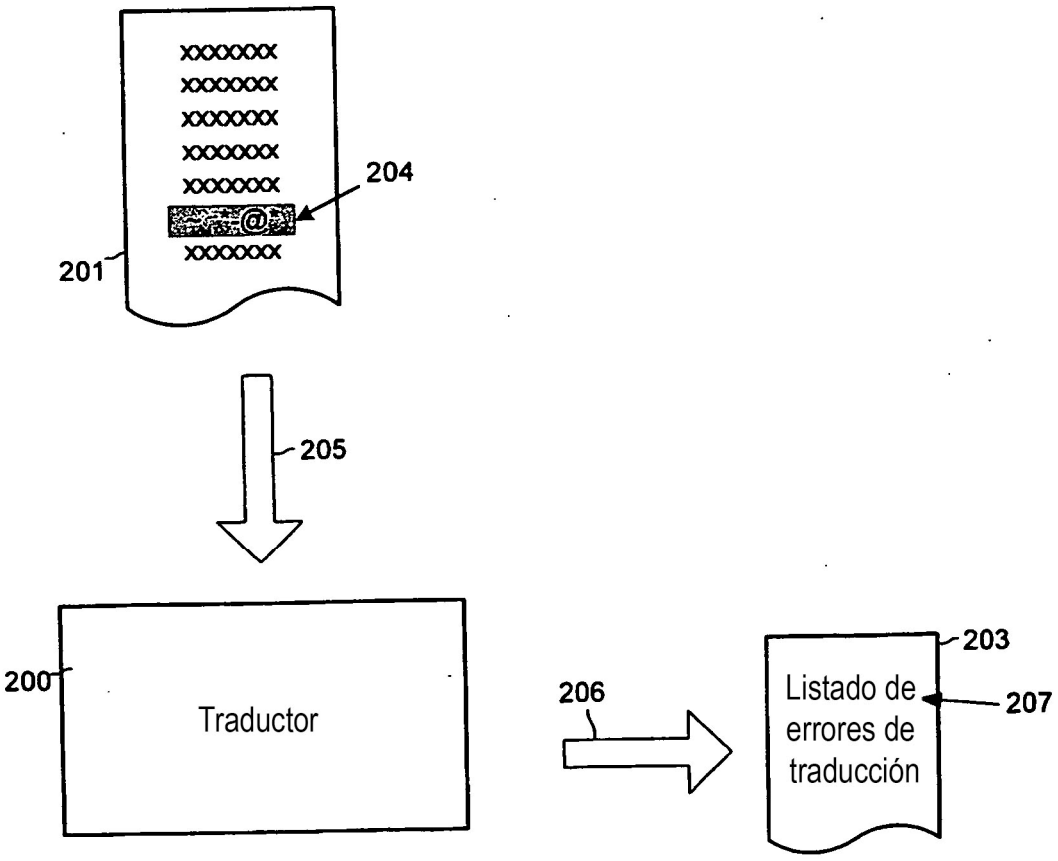


Fig. 3

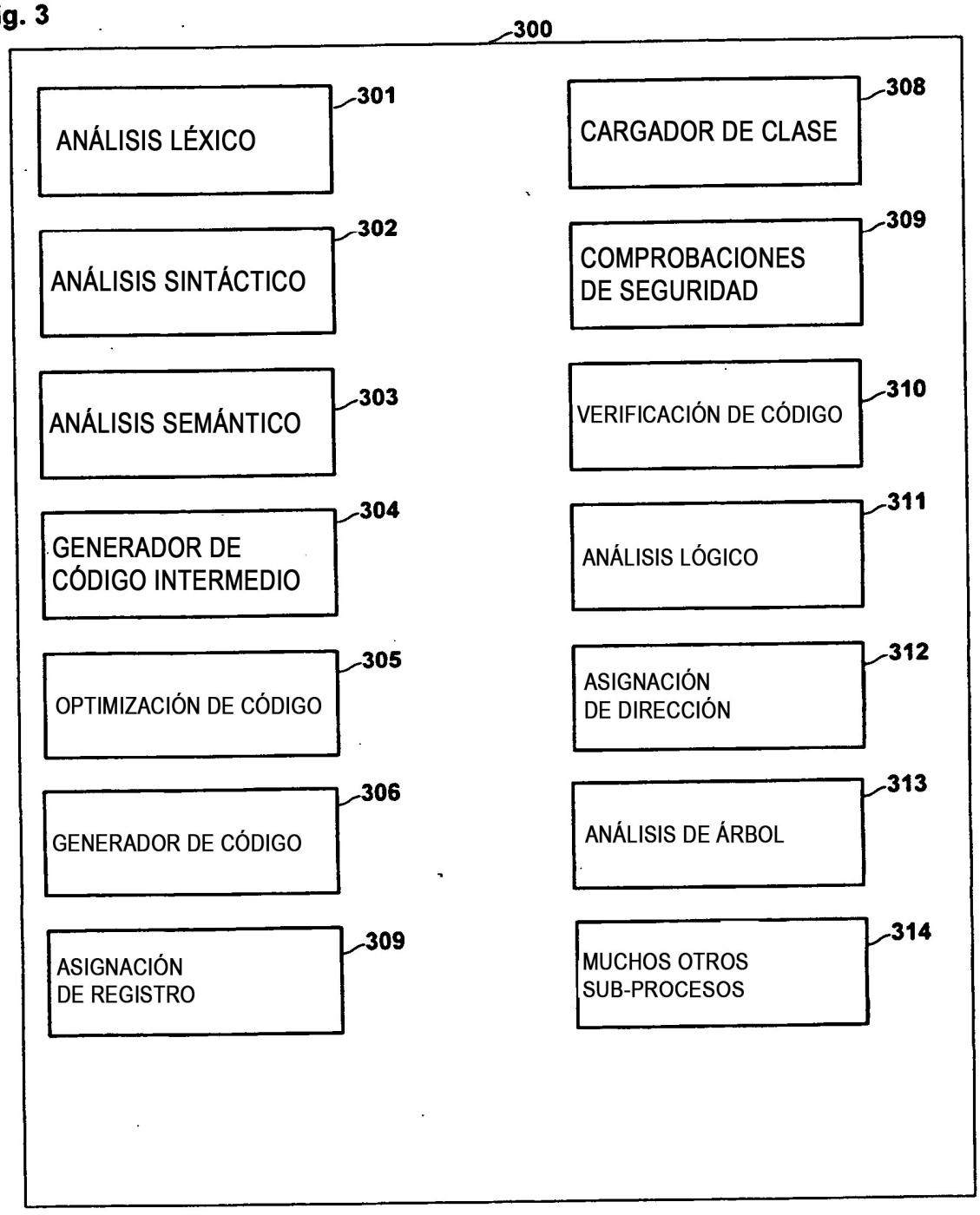

300
ANÁLISIS LÉXICO
301
ANÁLISIS SINTÁCTICO
302
ANÁLISIS SEMÁNTICO
303
GENERADOR DE CÓDIGO INTERMEDIO
304
OPTIMIZACIÓN DE CÓDIGO
305
GENERADOR DE CÓDIGO
306
ASIGNACIÓN DE REGISTRO
309
CARGADOR DE CLASE
308
COMPROBACIONES DE SEGURIDAD
309
VERIFICACIÓN DE CÓDIGO
310
ANÁLISIS LÓGICO
311
ASIGNACIÓN DE DIRECCIÓN
312
ANÁLISIS DE ÁRBOL
313
MUCHOS OTROS SUB-PROCESOS
314

Fig. 4

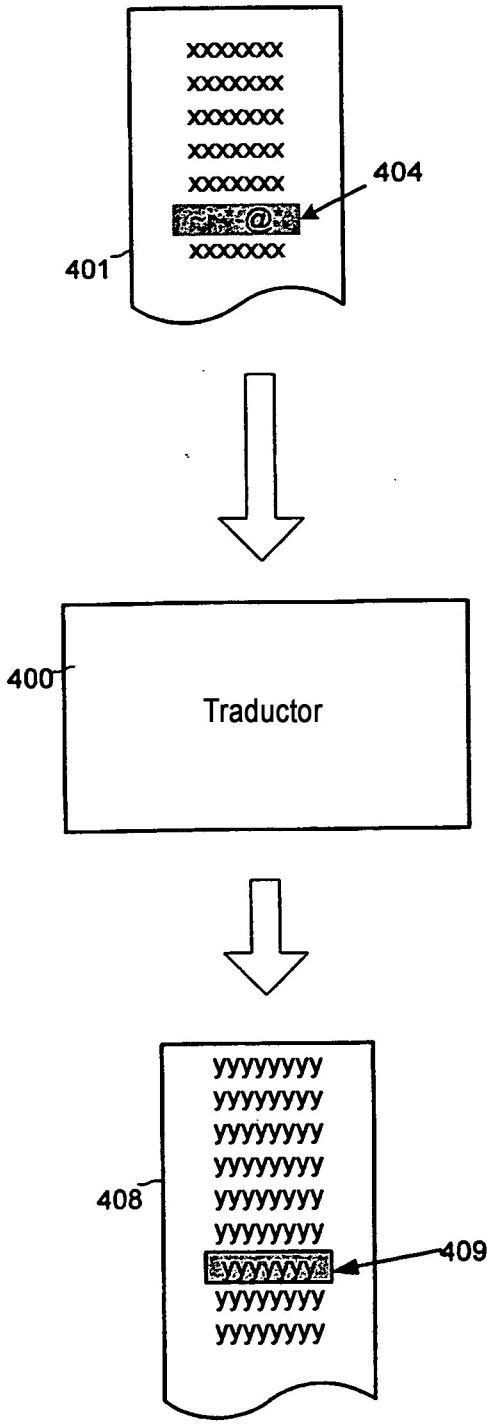

xxxxxxx
xxxxxxx
xxxxxxx
xxxxxxx
xxxxxxx
@
xxxxxxx
404
401
400
Traductor
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
408
409

Fig. 5

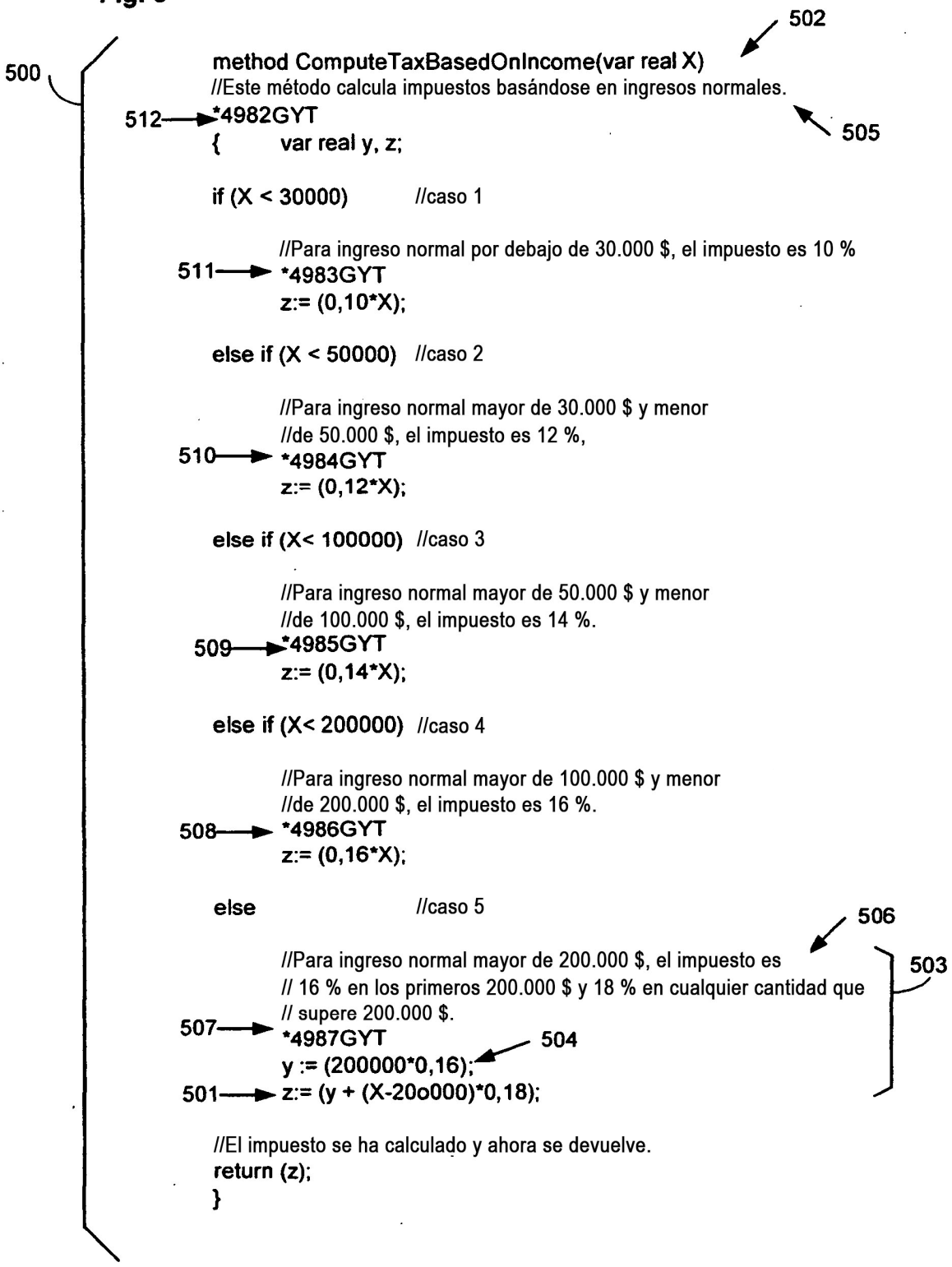

500
502
method ComputeTaxBasedOnIncome(var real X)
//Este método calcula impuestos basándose en ingresos normales.
512
*4982GYT
505
{ var real y, z;
if (X < 30000) //caso 1
//Para ingreso normal por debajo de 30.000 $, el impuesto es 10 %
511
*4983GYT
z:= (0,10*X);
else if (X < 50000) //caso 2
//Para ingreso normal mayor de 30.000 $ y menor
//de 50.000 $, el impuesto es 12 %,
510
*4984GYT
z:= (0,12*X);
else if (X< 100000) //caso 3
//Para ingreso normal mayor de 50.000 $ y menor
//de 100.000 $, el impuesto es 14 %.
509
*4985GYT
z:= (0,14*X);
else if (X< 200000) //caso 4
//Para ingreso normal mayor de 100.000 $ y menor
//de 200.000 $, el impuesto es 16 %.
508
*4986GYT
z:= (0,16*X);
else //caso 5
506
//Para ingreso normal mayor de 200.000 $, el impuesto es
// 16 % en los primeros 200.000 $ y 18 % en cualquier cantidad que
// supere 200.000 $.
503
507
*4987GYT
504
y := (200000*0,16);
501
z:= (y + (X-20o000)*0,18);
//El impuesto se ha calculado y ahora se devuelve.
return (z);
}

Fig. 6

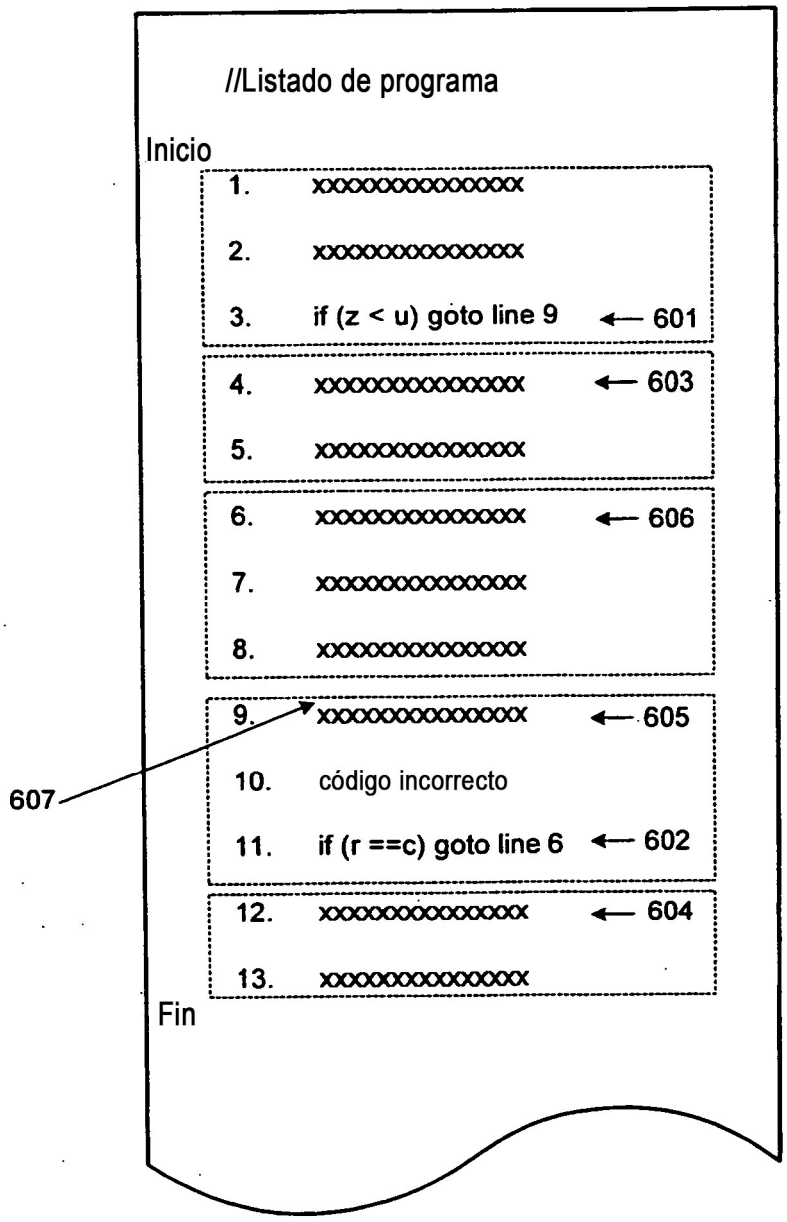

//Listado de programa
Inicio
1. xxxxxxxxxxxxxxx
2. xxxxxxxxxxxxxxx
3. if (z < u) goto line 9
601
4. xxxxxxxxxxxxxxx
603
5. xxxxxxxxxxxxxxx
6. xxxxxxxxxxxxxxx
606
7. xxxxxxxxxxxxxxx
8. xxxxxxxxxxxxxxx
9. xxxxxxxxxxxxxxx
605
607
10. código incorrecto
11. if (r ==c) goto line 6
602
12. xxxxxxxxxxxxxxx
604
13. xxxxxxxxxxxxxxx
Fin

Fig. 7

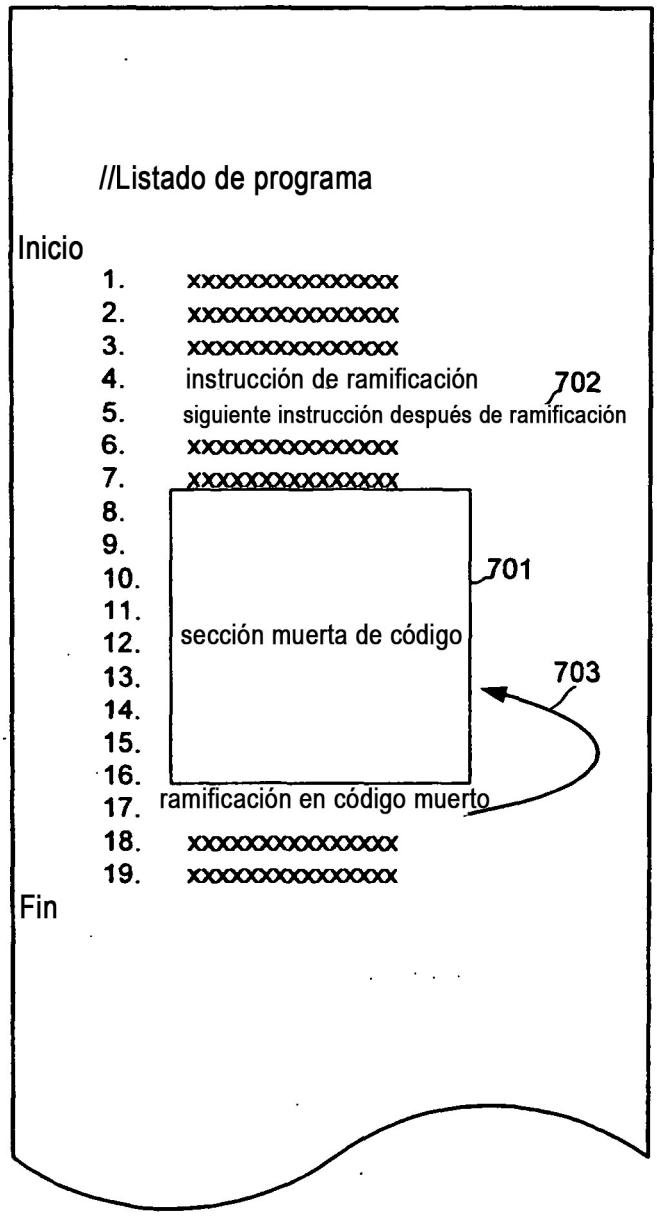

//Listado de programa
Inicio
1. xxxxxxxxxxxxxxx
2. xxxxxxxxxxxxxxx
3. xxxxxxxxxxxxxxx
4. instrucción de ramificación
5. siguiente instrucción después de ramificación
6. xxxxxxxxxxxxxxx
7. xxxxxxxxxxxxxxx
8.
9.
10.
11.
12. sección muerta de código
13.
14.
15.
16.
17. ramificación en código muerto
18. xxxxxxxxxxxxxxx
19. xxxxxxxxxxxxxxx
Fin
702
701
703

FIG. 8

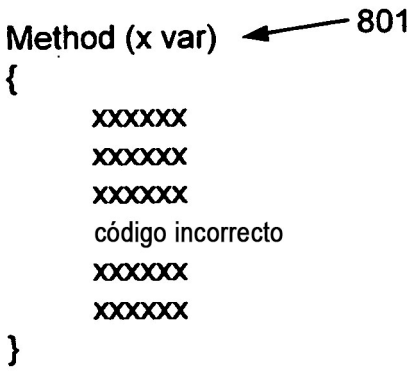

Method (x var)
801
{
xxxxxx
xxxxxx
xxxxxx
código incorrecto
xxxxxx
xxxxxx
}

Fig. 9

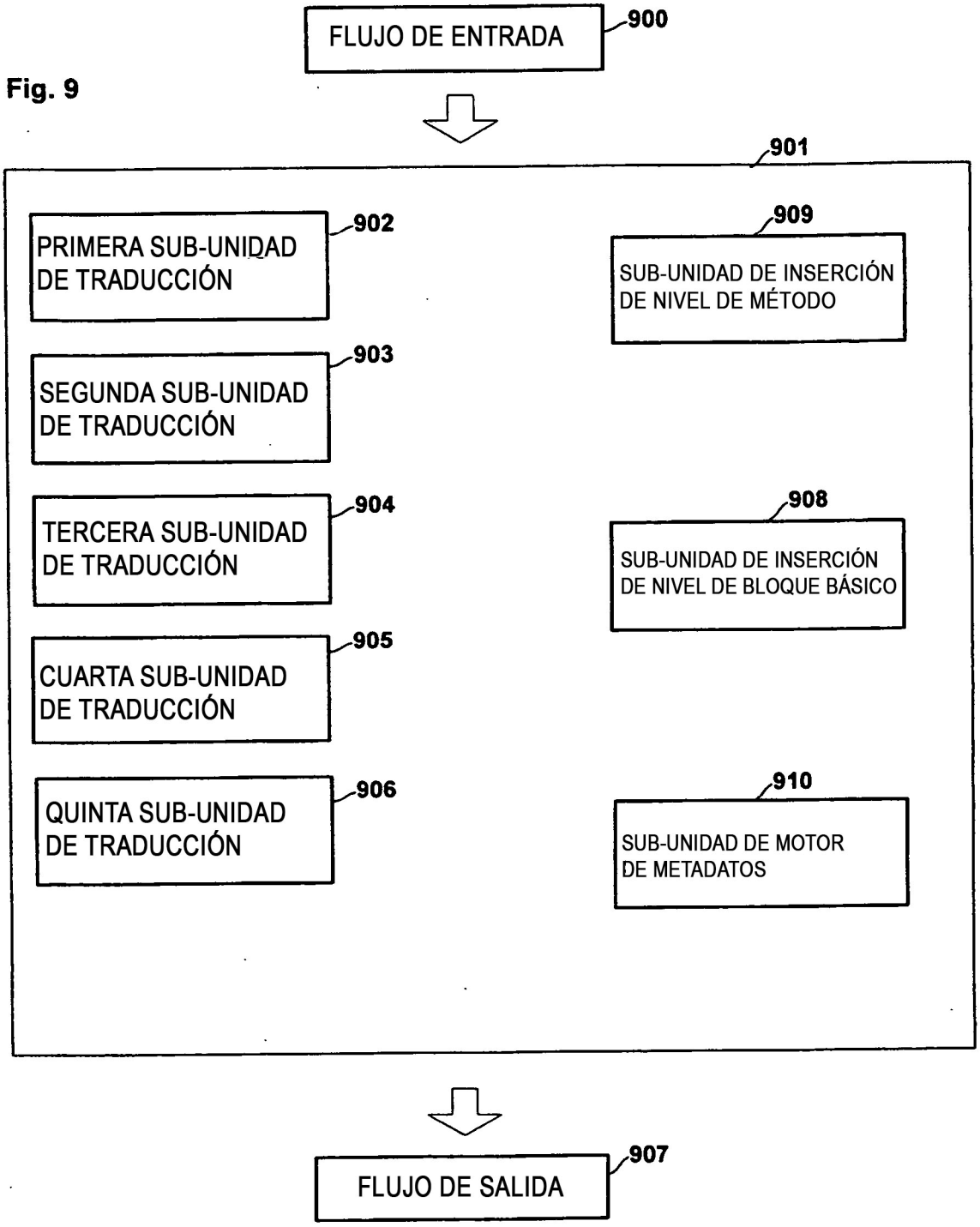

FLUJO DE ENTRADA
900
901
PRIMERA SUB-UNIDAD DE TRADUCCIÓN
902
SEGUNDA SUB-UNIDAD DE TRADUCCIÓN
903
TERCERA SUB-UNIDAD DE TRADUCCIÓN
904
CUARTA SUB-UNIDAD DE TRADUCCIÓN
905
QUINTA SUB-UNIDAD DE TRADUCCIÓN
906
SUB-UNIDAD DE INSERCIÓN DE NIVEL DE MÉTODO
909
SUB-UNIDAD DE INSERCIÓN DE NIVEL DE BLOQUE BÁSICO
908
SUB-UNIDAD DE MOTOR DE METADATOS
910
FLUJO DE SALIDA
907

Fig. 10

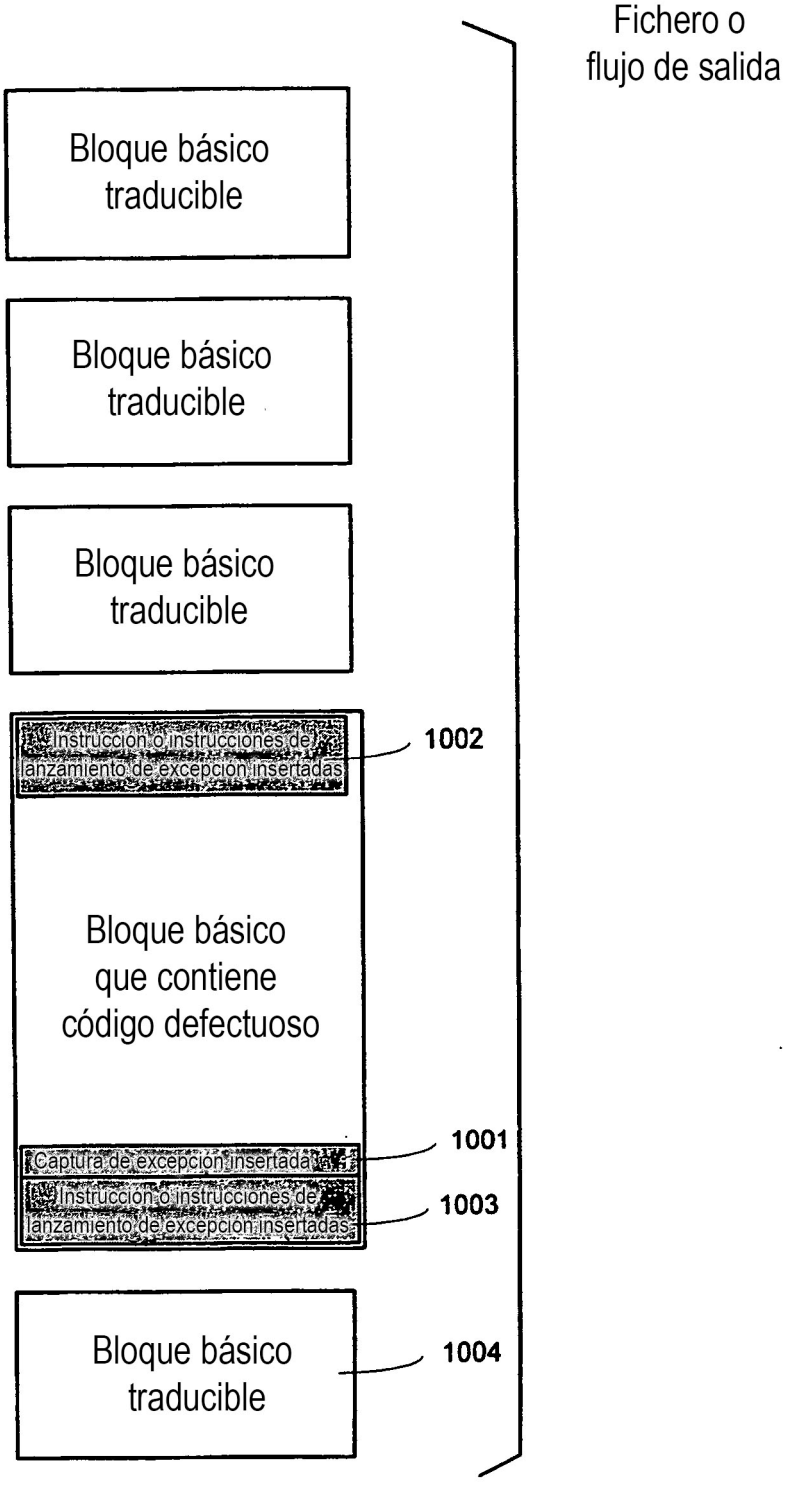
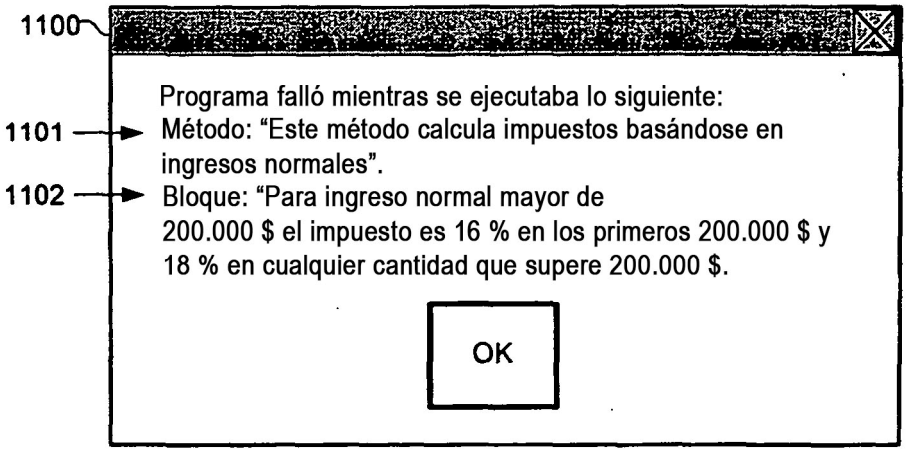

Fichero o flujo de salida
Bloque básico traducible
Bloque básico traducible
Bloque básico traducible
Instrucción o instrucciones de lanzamiento de excepción insertadas
1002
Bloque básico que contiene código defectuoso
1001
Captura de excepción insertada
Instrucción o instrucciones de lanzamiento de excepción insertadas
1003
Bloque básico traducible
1004

Fig. 11


1100
Programa falló mientras se ejecutaba lo siguiente:
1101
Método: "Este método calcula impuestos basándose en ingresos normales".
1102
Bloque: "Para ingreso normal mayor de 200.000 $ el impuesto es 16 % en los primeros 200.000 $ y 18 % en cualquier cantidad que supere 200.000 $.
OK

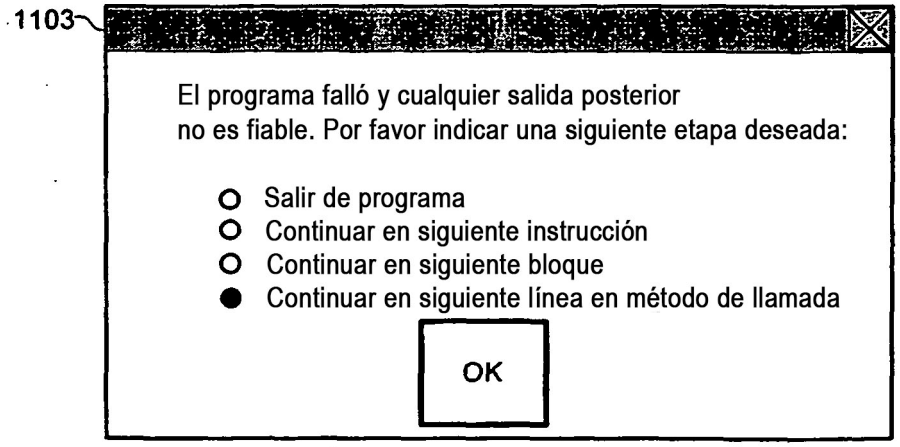

1103
El programa falló y cualquier salida posterior no es fiable. Por favor indicar una siguiente etapa deseada:
Salir de programa
Continuar en siguiente instrucción
Continuar en siguiente bloque
Continuar en siguiente línea en método de llamada
OK

Fig. 12

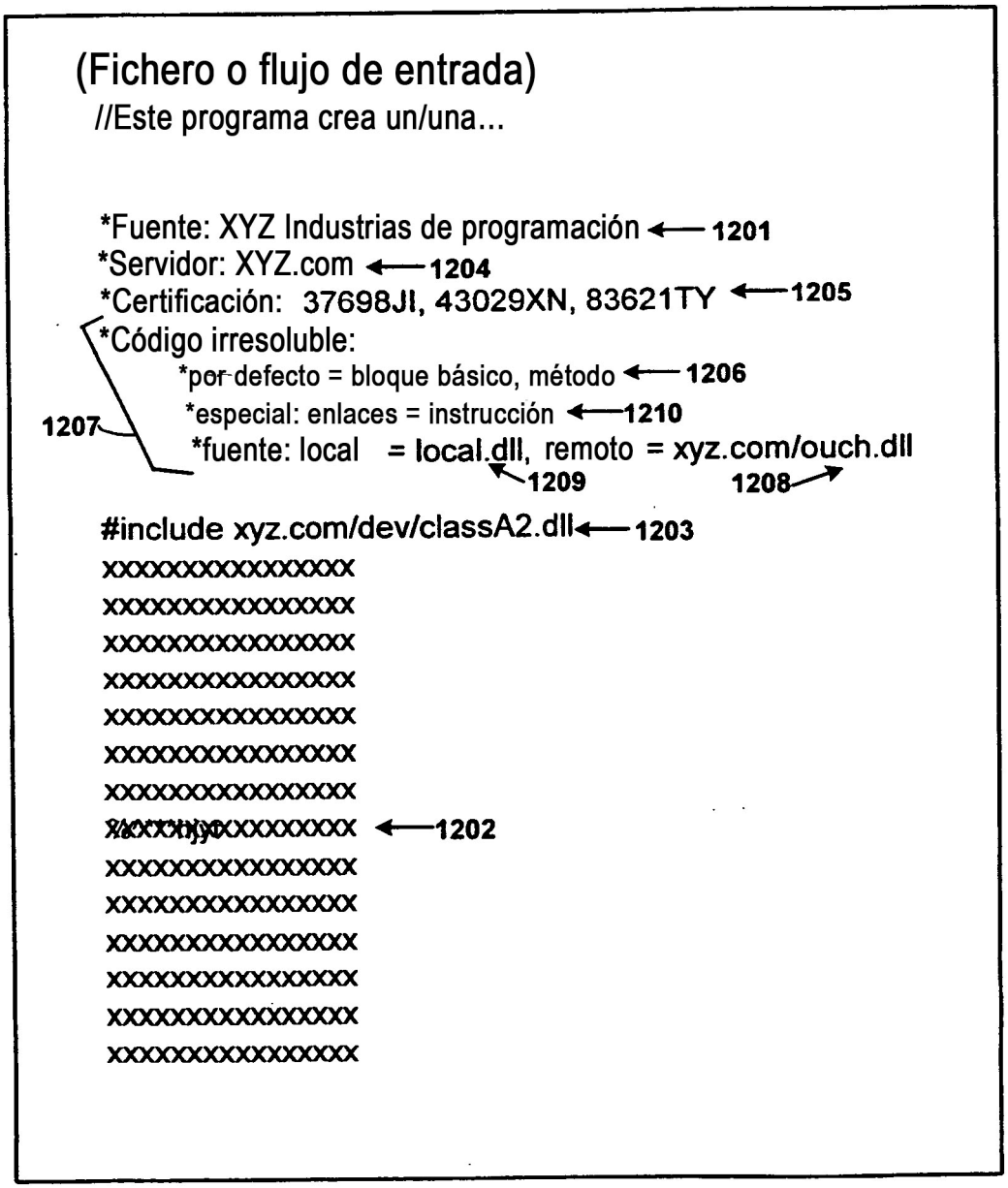

(Fichero o flujo de entrada)
//Este programa crea un/una...
*Fuente: XYZ Industrias de programación
1201
*Servidor: XYZ.com
1204
*Certificación: 37698JI, 43029XN, 83621TY
1205
*Código irresoluble:
*por defecto = bloque básico, método
1206
*especial: enlaces = instrucción
1210
1207
*fuente: local = local.dll, remoto = xyz.com/ouch.dll
1209
1208
#include xyz.com/dev/classA2.dll
1203
1202
Para todas las figuras:
'*' = metadatos
'//' = comentarios
'#' = enlace de tiempo de traducción

Fig. 13

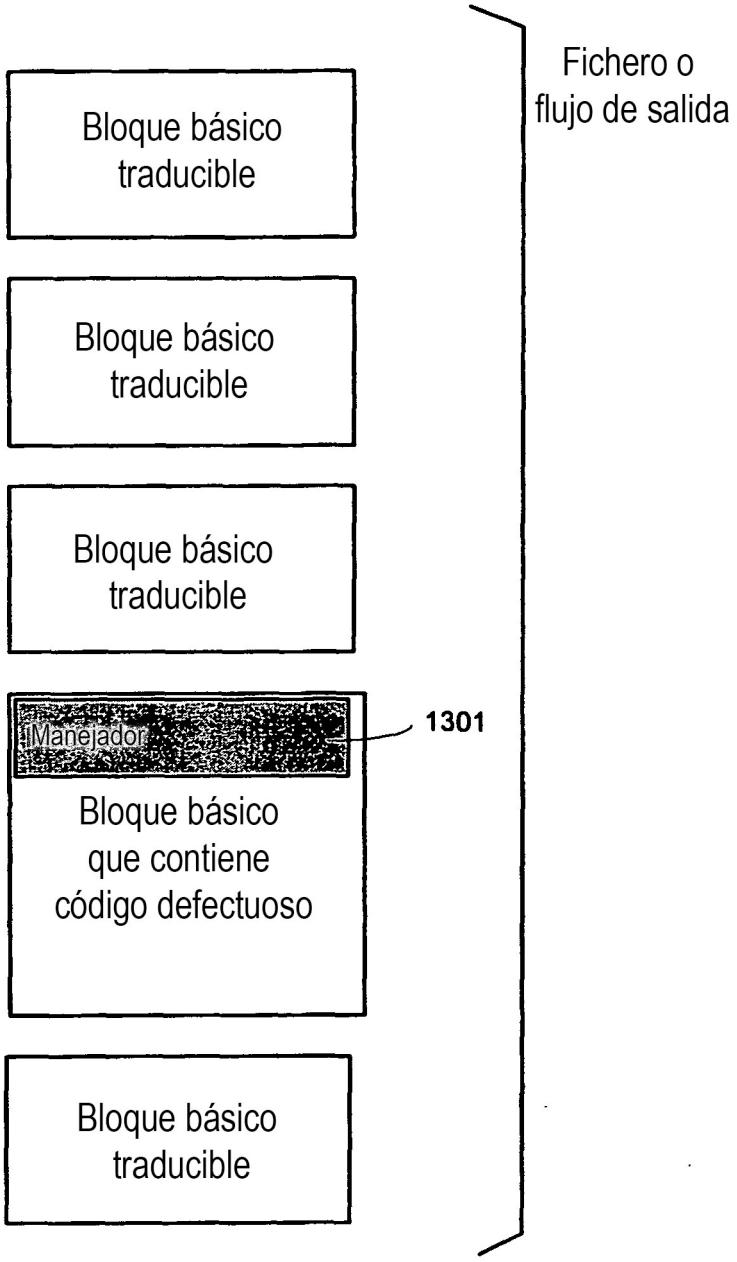

Fichero o
flujo de salida
Bloque básico
traducible
Bloque básico
traducible
Bloque básico
traducible
Manejador
1301
Bloque básico
que contiene
código defectuoso
Bloque básico
traducible

FIG. 14

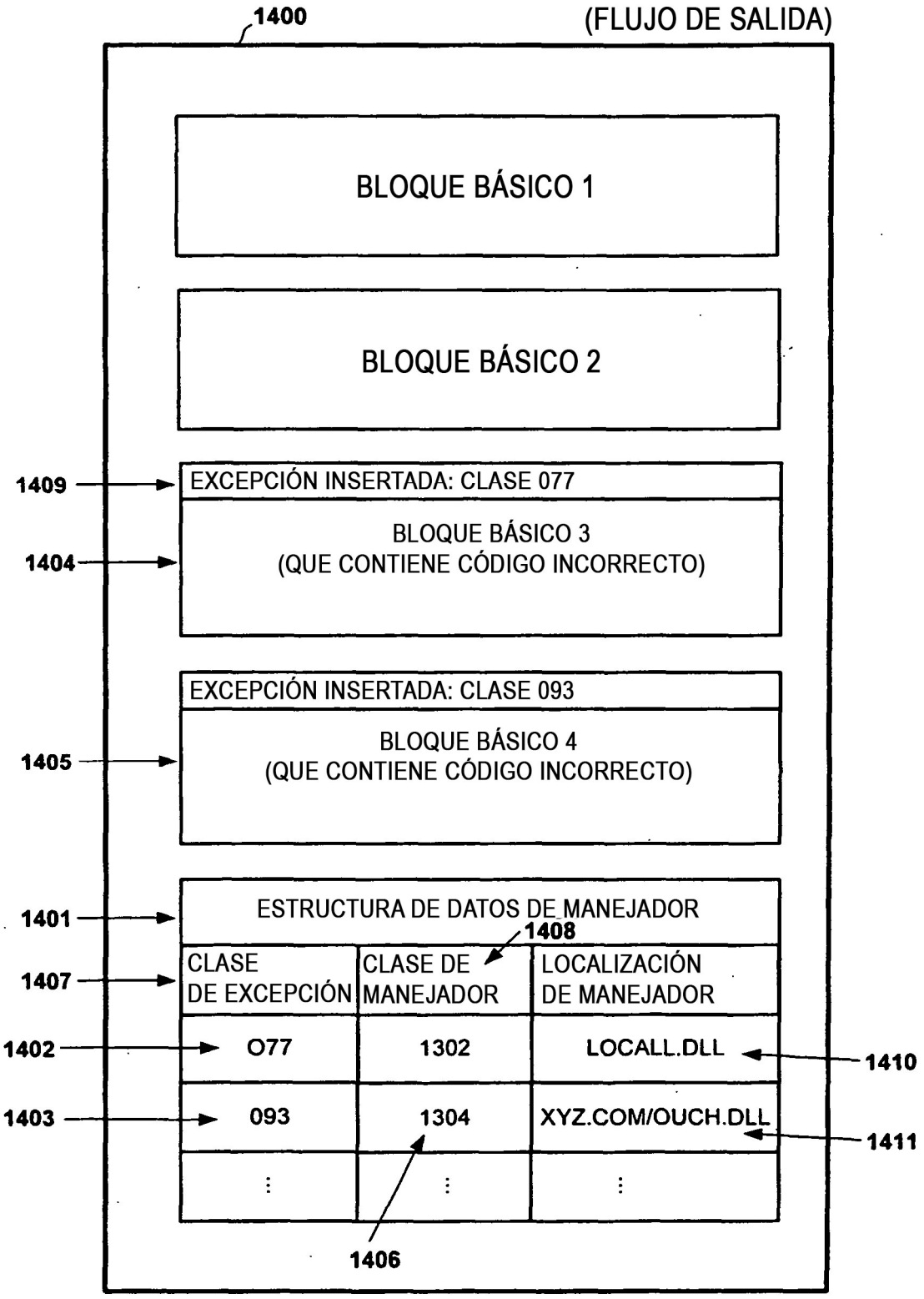

1400
(FLUJO DE SALIDA)
BLOQUE BÁSICO 1
BLOQUE BÁSICO 2
1409
EXCEPCIÓN INSERTADA: CLASE 077
1404
BLOQUE BÁSICO 3
(QUE CONTIENE CÓDIGO INCORRECTO)
EXCEPCIÓN INSERTADA: CLASE 093
1405
BLOQUE BÁSICO 4
(QUE CONTIENE CÓDIGO INCORRECTO)
1401
ESTRUCTURA DE DATOS DE MANEJADOR
1408
1407
CLASE DE EXCEPCIÓN
CLASE DE MANEJADOR
LOCALIZACIÓN DE MANEJADOR
1402
O77
1302
LOCALL.DLL
1410
1403
093
1304
XYZ.COM/OUCH.DLL
1411
1406

Fig. 15

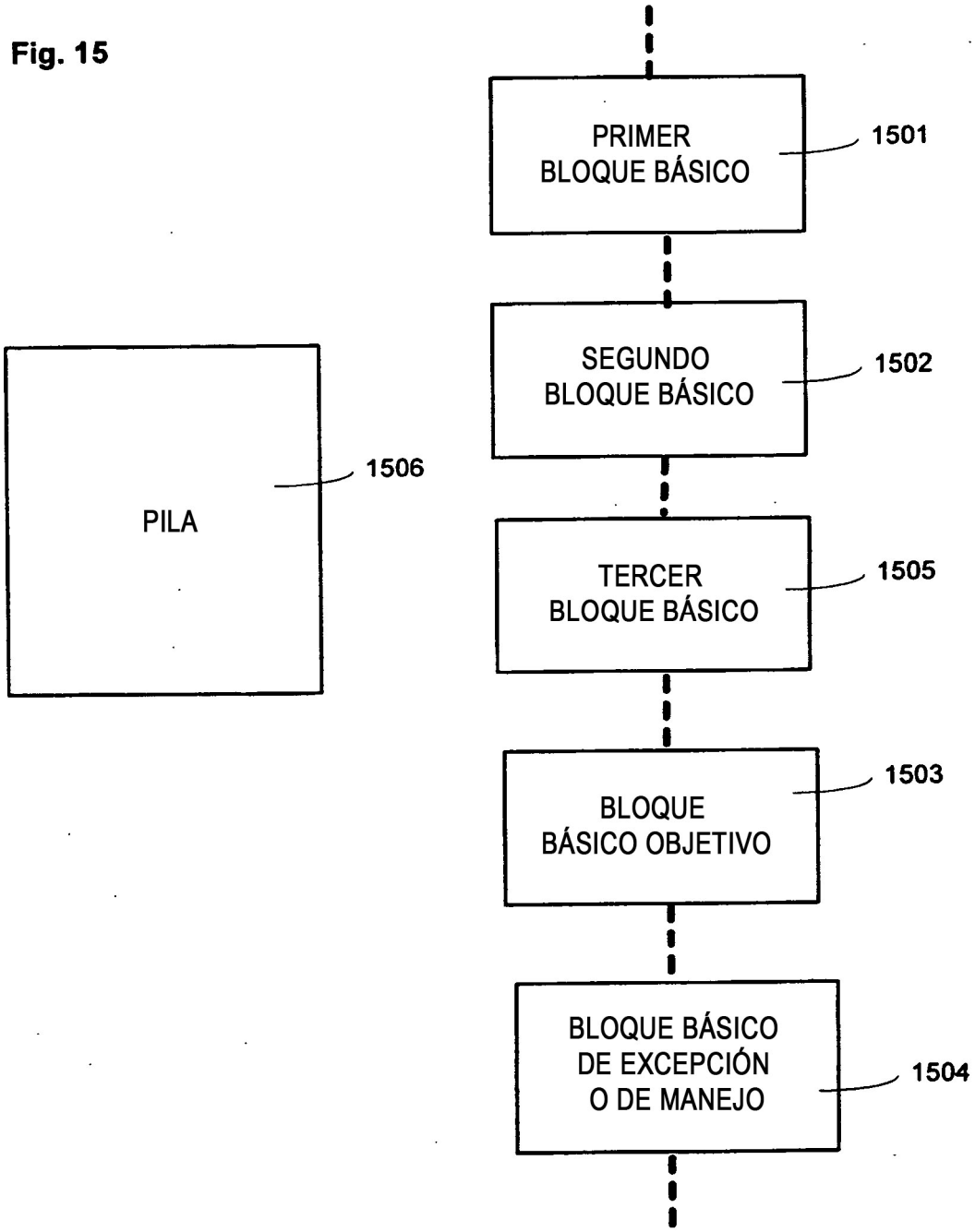

PRIMER
BLOQUE BÁSICO
1501
SEGUNDO
BLOQUE BÁSICO
1502
PILA
1506
TERCER
BLOQUE BÁSICO
1505
BLOQUE
BÁSICO OBJETIVO
1503
BLOQUE BÁSICO
DE EXCEPCIÓN
O DE MANEJO
1504

Fig. 16

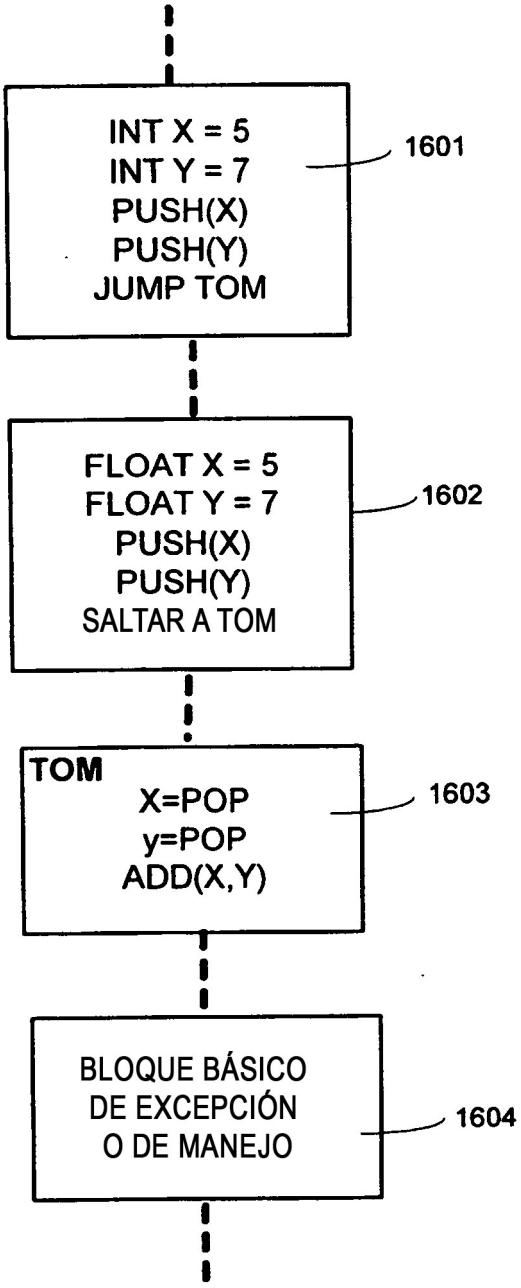
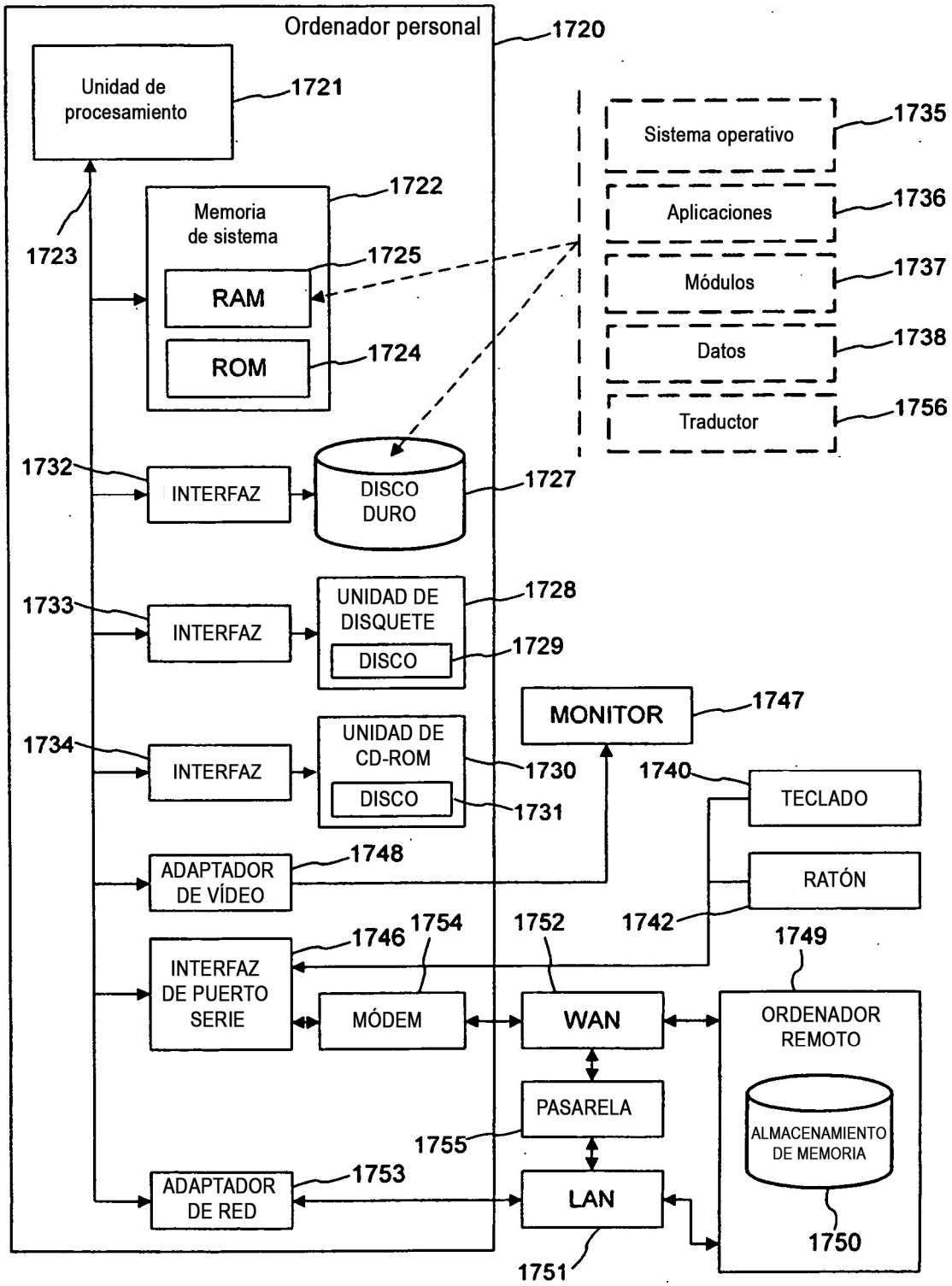

INT X = 5
INT Y = 7
PUSH(X)
PUSH(Y)
JUMP TOM
1601
FLOAT X = 5
FLOAT Y = 7
PUSH(X)
PUSH(Y)
SALTAR A TOM
1602
TOM
X=POP
y=POP
ADD(X,Y)
1603
BLOQUE BÁSICO
DE EXCEPCIÓN
O DE MANEJO
1604

Fig. 17


Ordenador personal
1720
Unidad de procesamiento
1721
1723
Memoria de sistema
1722
RAM
1725
ROM
1724
Sistema operativo
1735
Aplicaciones
1736
Módulos
1737
Datos
1738
Traductor
1756
1732
INTERFAZ
DISCO DURO
1727
1733
INTERFAZ
UNIDAD DE DISQUETE
DISCO
1728
1729
1734
INTERFAZ
UNIDAD DE CD-ROM
DISCO
1730
1731
MONITOR
1747
1740
TECLADO
RATÓN
1742
ADAPTADOR DE VÍDEO
1748
INTERFAZ DE PUERTO SERIE
1746
1754
MÓDEM
1752
WAN
1749
ORDENADOR REMOTO
ALMACENAMIENTO DE MEMORIA
1750
PASARELA
1755
ADAPTADOR DE RED
1753
LAN
1751