(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 600 06 141 T2** 2004.08.26

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 212 678 B1**
(21) Deutsches Aktenzeichen: **600 06 141.8**
(86) PCT-Aktenzeichen: **PCT/FR00/02349**
(96) Europäisches Aktenzeichen: **00 958 714.8**
(87) PCT-Veröffentlichungs-Nr.: **WO 01/014958**
(86) PCT-Anmeldetag: **21.08.2000**
(87) Veröffentlichungstag der PCT-Anmeldung: **01.03.2001**
(97) Erstveröffentlichung durch das EPA: **12.06.2002**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **22.10.2003**
(47) Veröffentlichungstag im Patentblatt: **26.08.2004**

(51) Int Cl.[7]: **G06F 9/445**
G06F 9/45

(30) Unionspriorität:
9910697 **23.08.1999** **FR**

(84) Benannte Vertragsstaaten:
AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LI, LU, MC, NL, PT, SE

(73) Patentinhaber:
Trusted Logic, Versailles, FR

(72) Erfinder:
LEROY, Xavier, F-78000 Versailles, FR

(74) Vertreter:
Heuer, W., Dipl.-Phys. Dr.rer.nat., Pat.-Anw., 82515 Wolfratshausen

(54) Bezeichnung: **VERWALTUNGSPROTOKOLL, VERIFIKATIONSVERFAHREN UND TRANSFORMIERUNG EINES FERNGELADENEN PROGRAMMFRAGMENTS UND KORRESPONDIERENDE SYSTEME**

Beschreibung

[0001] Die Erfindung betrifft ein Verfahren zur Verwaltung, ein Verfahren zur Überprüfung, ein Verfahren zur Umwandlung eines ferngeladenen Programmfragments und die entsprechenden Systeme, genauer gesagt solche, die für mitgeführte Informatiksysteme bestimmt sind, die über geringe Ressourcen an Speicher und an Rechenleistung verfügen.

[0002] Allgemein umfassen mit Bezug auf **Fig.** 1A mitgeführte Informatiksysteme **10** einen Mikroprozessor **11**, einen Permanentspeicher wie etwa einen nicht beschreibbaren Speicher **12**, der den Code des ausführbaren Programms enthält, einen überschreibbaren, nichtflüchtigen, permanenten Speicher **13** vom Typ EEPROM, der die in dem System gespeicherten Daten enthält, einen flüchtigen Schreiblesespeicher **14**, in welchem das Programm seine Zwischenergebnisse während seiner Ausführung speichert, und Ein-/Ausgabevorrichtungen **15**, die es dem System ermöglichen, mit seiner Umgebung zu wechselwirken. In dem Fall, wo das mitgeführte Informatiksystem durch eine Mikroprozessorkarte vom Typ Geldkarte oder Scheckkarte gebildet ist, besteht die Ein-/Ausgabevorrichtung **15** aus einer seriellen Verbindung, die es der Karte ermöglicht, mit einem Endgerät wie etwa einem Kartenlesegerät zu kommunizieren.

[0003] Bei den herkömmlichen mitgeführten Informatiksystemen wird der Code des von dem System ausgeführten Programms beim Bau des Systems oder spätestens bei der Personalisierung des letzteren vor Auslieferung an den Endbenutzer festgelegt.

[0004] Es sind weiterentwickelte mitgeführte Informatiksysteme eingesetzt worden, die reprogrammierbar sind, wie z. B. die Mikroprozessorkarten vom Typ JavaCard. Diese umprogrammierbaren Systeme bieten gegenüber den vorhergehenden die zusätzliche Möglichkeit, das Programm nach Inbetriebnahme des Systems durch eine Operation des Fernladens von Programmfragmenten anzureichern. Diese Programmfragmente, im Englischen allgemein als „Applets" bezeichnet, werden in der vorliegenden Beschreibung unterschiedslos als Applets oder Programmfragmente bezeichnet. Für eine detailliertere Beschreibung der JavaCard-Systeme wird auf die von der Fa. Sun Microsystems Inc. herausgegebene Dokumentation verwiesen, insbesondere auf die elektronisch verfügbare Dokumentation, Kapitel JavaCard technology auf der Seite W.W.W. (World Wide Web) http://java.sun.com/products/javacard/index.html, Juni 1999.

[0005] **Fig.** 1b zeigt die Architektur eines solchen umprogrammierbaren mitgeführten Informatiksystems. Diese Architektur ist ähnlich derjenigen eines herkömmlichen mitgeführten Systems, mit dem Unterschied, dass das umprogrammierbare mitgeführte System ferner Applets über eine seiner Ein-/Ausgabevorrichtungen empfangen und diese dann in seinem Permanentspeicher **13** speichern kann, von wo aus sie anschließend in Ergänzung des Hauptprogramms ausgeführt werden können.

[0006] Aus Gründen der Portabilität zwischen verschiedenen mitgeführten Informatiksystemen liegen die Applets in Form von Code für eine virtuelle Standardmaschine vor. Dieser Code ist nicht direkt vom Mikroprozessor **11** ausführbar, sondern muss softwaremäßig durch eine virtuelle Maschine **16** interpretiert werden, die durch ein Programm gebildet ist, das im nicht beschreibbaren Permanentspeicher **12** vorliegt. In dem erwähnten Beispiel der JavaCard-Karten ist die verwendete virtuelle Maschine eine Untermenge der virtuellen Java-Maschine. Für eine Beschreibung der die virtuelle Java-Maschine und die verwendete virtuelle Maschine betreffenden Spezifikationen wird auf das von Tim Lindholm und Frank Yellin veröffentlichte Werk mit dem Titel „The Java Virtual Machine Specification", Addison-Wesley 1996 und die von der Fa. Sun Microsystems Inc. herausgegebene Dokumentation „JavaCard 2.1 Virtual Machine Specification" verwiesen, die auf der Seite W.W.W.http://java.sun.com/products/javacard/JCVMSpec.pdf, März 1999, elektronisch verfügbar ist.

[0007] Die Operation des Fernladens von Applets in ein mitgeführtes Informatiksystem im Betrieb wirft erhebliche Sicherheitsprobleme auf. Ein versehentlich oder gar absichtlich schlecht geschriebenes Applet kann die in dem System vorhandenen Daten fehlerhaft abändern, das Hauptprogramm daran hindern, korrekt oder zur gewollten Zeit abzulaufen, oder auch andere zuvor ferngeladene Applets verändern, wodurch diese unbrauchbar oder schädlich werden.

[0008] Ein von einem Cyberpiraten geschriebenes Applet kann sogar in dem System gespeicherte vertrauliche Informationen verbreiten, wie etwa im Fall einer Scheckkarte z. B. den Zugangscode.

[0009] Gegenwärtig sind drei Lösungen vorgeschlagen worden, um das Problem der Sicherheit der Applets zu lösen.

[0010] Eine erste Lösung beruht in der Verwendung von kryptographischen Unterschriften, so dass nur Applets akzeptiert werden, die von vertrauenswürdigen Personen oder Organisationen herrühren.

[0011] Bei dem genannten Beispiel einer Scheckkarte werden nur die Applets, die die kryptographische Unterschrift der Bank tragen, die die Karte ausgegeben hat, von der Karte akzeptiert und ausgeführt, und jedes andere nicht unterschriebene Applet wird im Laufe der Fernladeoperation zurückgewiesen. Ein böswilliger Benutzer der Karte, der nicht über den Verschlüsselungsschlüssel der Bank verfügt, ist also nicht in der Lage, ein nicht unterschriebenes und gefährliches Applet auf der Karte ausführen zu lassen.

[0012] Diese erste Lösung ist gut geeignet für den Fall, wo alle Applets von einer gleichen, einzigen Quelle, im genannten Beispiel der Bank, kommen. Diese Lösung ist schwerlich anwendbar in dem Fall, wo die Applets

von mehreren Quellen kommen, wie etwa, bei dem Beispiel einer Scheckkarte, der Hersteller der Karte, die Bank, die Organisationen, die Dienstleistungen über Scheckkarte verwalten, die großen Handelsorganisationen, die der Kundschaft Kundenbindungsprogramme anbieten und legitimerweise anbieten, spezielle Applets auf die Karte zu laden. Gemeinsame Nutzung und gemeinsamer Besitz der für die elektronische Unterzeichnung der Applets notwendigen Verschlüsselungsschlüssel durch diese unterschiedlichen Wirtschaftsteilnehmer werfen erhebliche technische, wirtschaftliche und juristische Probleme auf.

[0013] Eine zweite Lösung beruht darin, dynamische Kontrollen des Zugangs und der Typisierung während der Ausführung der Applets durchzuführen.

[0014] Bei dieser Lösung führt die virtuelle Maschine bei Ausführung der Applets eine bestimmte Zahl von Kontrollen durch, wie etwa:

– Kontrolle des Zugangs zum Speicher: bei jedem Lesen oder Schreiben einer Speicherzone überprüft die virtuelle Maschine das Recht des Applets, auf die entsprechenden Daten zuzugreifen;

– dynamische Überprüfung der Datentypen: bei jedem Befehl des Applets überprüft die virtuelle Maschine, dass die Anforderungen an die Datentypen erfüllt sind. Z. B. kann die virtuelle Maschine Daten wie etwa gültige Speicheradressen gesondert behandeln und verhindern, dass das Applet ungültige Speicheradressen durch Ganzzahlig-/Adresse-Konversionen oder durch arithmetische Operationen an Adressen erzeugt;

– Erfassung des Stacküberlaufs und von illegalen Zugriffen auf den Ausführungsstack der virtuellen Maschine, die unter bestimmten Bedingungen deren Funktion so stören können, dass die oben genannten Kontrollmechanismen umgangen werden.

[0015] Diese zweite Lösung erlaubt die Ausführung eines großen Spektrums von Applets unter befriedigenden Sicherheitsbedingungen. Sie hat allerdings den Nachteil einer beträchtlichen Verlangsamung der Ausführung, die durch die Menge der dynamischen Überprüfungen hervorgerufen wird. Um eine Verringerung dieser Verlangsamung zu erreichen, kann ein Teil dieser Überprüfungen von dem Mikroprozessor selbst übernommen werden, auf Kosten einer Zunahme der Komplexität des letzteren und damit der Gestehungskosten des mitgeführten Systems. Solche Überprüfungen erhöhen außerdem den Bedarf an Schreiblesespeicher und Permanentspeicher des Systems aufgrund der zusätzlichen Typinformationen, die den gehandhabten Daten zugeordnet werden müssen.

[0016] Eine dritte Lösung beruht darin, eine statische Überprüfung des Codes des Applets beim Fernladen durchzuführen.

[0017] Bei dieser Lösung simuliert diese statische Überprüfung die Ausführung des Applets auf dem Niveau der Datentypen und stellt ein für allemal fest, dass der Code des Applets die Regeln für Datentypen und Zugangskontrolle, wie von der virtuellen Maschine vorgegeben, erfüllt und keinen Stacküberlauf hervorruft. Wenn diese statische Überprüfung gelingt, kann das Applet anschließend ausgeführt werden, ohne dass es erforderlich wird, dynamisch zu überprüfen, dass die Regel eingehalten wird. In dem Fall, wo der Prozess der statischen Überprüfung scheitert, verwirft das mitgeführte System das Applet und erlaubt dessen spätere Ausführung nicht. Für eine genauere Beschreibung der dritten genannten Lösung kann auf das oben zitierte Werk von Tim Lindholm und Frank Yellin, auf den Artikel von James A. Gosling mit dem Title „Java Intermediate Byte Codes", Verhandlungen der ACM SIGPLAN, Workshop on Intermediate Representations (IR'95), Seiten 111–118, Januar 1995 und das US Patent 5,748,964, erteilt am 5.5.1998, zurückgegriffen werden.

[0018] Gegenüber der zweiten Lösung hat die dritte Lösung den Vorteil einer wesentlichen schnelleren Ausführung der Applets, weil die virtuelle Maschine während der Ausführung keine Überprüfung durchführt.

[0019] Die dritte Lösung hat jedoch den Nachteil eines statischen Codeüberprüfungsprozesses, der kompliziert und kostspielig ist, sowohl hinsichtlich der zur Durchführung des Prozesses erforderlichen Codegröße wie auch in Bezug auf die Größe des Schreiblesespeichers, der notwendig ist, um die Zwischenergebnisse der Überprüfung aufzunehmen, wie auch hinsichtlich der Rechenzeit. Als verdeutlichendes Beispiel stellt die in das von Sun Microsystems vertriebene System Java JDK, integrierte Codeüberprüfung in der Größenordnung von 50 kByte Maschinencode dar, und sein Verbrauch an Schreiblesespeicher ist proportional zu (Tp + Tr) × Nb, wobei Tp den maximalen Stackraum, Tr die maximale Zahl von Registern und Nb die maximale Zahl von von einem Unterprogramm, allgemein auch als Methode bezeichnet, verwendeten Sprungzielen des Applets bezeichnet. Dieser Speicherbedarf übertrifft bei weitem die Kapazitäten der Ressourcen der Mehrheit der gegenwärtigen mitgeführten Informatiksysteme, insbesondere der kommerziell verfügbaren Mikroprozessorkarten.

[0020] Mehrere Varianten der dritten Lösung sind vorgeschlagen worden, bei denen der Herausgeber des Applets an den Überprüfer neben dem Code des Applets eine bestimmte Zahl von zusätzlichen spezifischen Informationen überträgt, wie etwa vorberechnete Datentypen oder ein Vorabbeweis einer korrekten Datentypisierung. Für eine detailliertere Beschreibung der entsprechenden Betriebsmodi kann auf die Artikel von Eva Rose und Kristoffer Høgsbro Rose, „Lightweight Bytecode Verification", Verhandlungen des Workshop Formal Underspinning of Java, Oktober 1998 und von George C. Necula „Proof-Carrying Code", Verhandlungen des 24. ACM Symposium Principles of Programming Languages, Seiten 106–119, zurückgegriffen werden.

[0021] Diese zusätzlichen Informationen erlauben es, den Code schneller zu überprüfen und die Größe des

Überprüfungsprogrammcodes etwas zu verringern, sie ermöglichen aber keine Verringerung des Bedarfs an Schreiblesespeicher oder vergrößern diesen sogar in erheblichem Maß im Fall der voretablierten Beweisinformationen für eine korrekte Datentypisierung.

[0022] Ziel der vorliegenden Erfindung ist, die erwähnten Nachteile des Standes der Technik zu beheben.

[0023] Insbesondere ist ein Gegenstand der vorliegenden Erfindung die Anwendung eines Protokolls zur Verwaltung eines ferngeladenen Programmfragments oder Applets, das eine Ausführung des letzteren durch ein mitgeführtes Informatiksystem ermöglicht, das über geringe Ressourcen verfügt, wie etwa eine Mikroprozessorkarte.

[0024] Ein anderer Gegenstand der Erfindung ist ebenfalls die Anwendung eines Verfahrens zur Überprüfung eines ferngeladenen Programmfragments oder Applets, bei dem ein Prozess zur statischen Überprüfung des Codes des Applets bei dessen Fernladen durchgeführt wird, wobei dieser Prozess wenigstens dem Prinzip nach der dritten oben beschriebenen Lösung angenähert werden kann, bei dem aber neuartige Techniken zur Überprüfung eingesetzt werden, um die Ausführung dieser Überprüfung mit den Werten der Speichergröße und der Rechengeschwindigkeit zu ermöglichen, die von den Mikroprozessorkarten und anderen mitgeführten Informatiksystemen geringer Leistung vorgegeben sind.

[0025] Ein anderes Ziel der vorliegenden Erfindung ist ebenfalls die Anwendung von Verfahren zum Umwandeln von Programmfragmenten herkömmlichen Typs, die z. B. durch Anwendung eines Java-Compilers erhalten sind, in normierte Programmfragmente oder Applets, die a priori den Kriterien der Überprüfung des erfindungsgegenständlichen Überprüfungsverfahrens genügen, um den Prozess der Überprüfung und Ausführung dieser letzteren an den mitgeführten Informatiksystemen oder gegenwärtigen Mikroprozessorkarten zu beschleunigen.

[0026] Ein anderes Ziel der vorliegenden Erfindung ist schließlich die Realisierung von mitgeführten Informatiksystemen, die die Anwendung des oben erwähnten Protokolls zur Verwaltung und des Verfahrens zur Überprüfung eines ferngeladenen Programmfragments ermöglichen, sowie von Informatiksystemen, die die Anwendung von Verfahren zur Umwandlung von herkömmlichen Programmfragmenten oder Applets in die erwähnten normierten Programmfragmente oder Applets ermöglichen.

[0027] Das erfindungsgegenständliche Protokoll zur Verwaltung eines ferngeladenen Programmfragments auf einem umprogrammierbaren mitgeführten System ist insbesondere anwendbar auf eine Mikroprozessorkarte, die mit einem überschreibbaren Speicher ausgestattet ist. Das Programmfragment ist gebildet durch einen Objektcode, eine Folge von Anweisungen, die durch den Mikroprozessor des mitgeführten Systems über eine virtuelle Maschine ausführbar ist, die mit einem Ausführungsstack und Registern oder lokalen Variablen ausgestattet ist, die von diesen Anweisungen manipuliert werden, und die es erlaubt, diesen Objektcode zu interpretieren. Das mitgeführte System ist an ein Endgerät angeschlossen.

[0028] Es ist bemerkenswert, dass es, wenigstens auf dem Niveau des mitgeführten Systems, darin beruht, einen Befehl zum Fernladen des Programmfragments zu erfassen. Bei positiver Reaktion auf den Schritt, der darin beruht, einen Fernladebefehl zu erfassen, beruht es ferner darin, den das Programmfragment bildenden Objektcode zu lesen und diesen Objektcode zeitweilig in dem überschreibbaren Speicher zu speichern. Der gesamte gespeicherte Objektcode wird Anweisung für Anweisung einem Überprüfungsprozess unterzogen. Der Überprüfungsprozess beruht in wenigstens einem Schritt der Initialisierung des Stacks der Typen und der Tabelle der Typen von Registern, die den Zustand der virtuellen Maschine zu Beginn der Ausführung des zeitweilig gespeicherten Objektcodes darstellen, und in einer Folge von Schritten der Überprüfung, Anweisung für Anweisung, der Existenz, für jede laufende Anweisung, eines Zieles, eines Zieles einer Verzweigungsanweisung, eines Zieles eines Ausnahmeverwaltungsaufrufes, und einer Überprüfung und einer Aktualisierung der Wirkung der laufenden Anweisung auf den Stack der Typen und die Tabelle der Registertypen. Im Fall einer nicht gelungenen Überprüfung des Objektcodes beruht das erfindungsgegenständliche Protokoll darin, das zeitweilig aufgezeichnete Programmfragment zu löschen, ohne dass eine Aufzeichnung des letzteren in dem Verzeichnis verfügbarer Programme aufgezeichnet ist, und einen Fehlercode an das Lesegerät zu senden.

[0029] Das Verfahren zur Überprüfung eines ferngeladenen Programmfragments in einem mitgeführten System gemäß der Erfindung ist insbesondere anwendbar auf eine mit einem überschreibbaren Speicher versehene Mikroprozessorkarte. Das Programmfragment ist gebildet durch einen Objektcode und umfasst wenigstens ein Unterprogramm, eine Folge von Anweisungen, die durch den Mikroprozessor des mitgeführten Systems über eine virtuelle Maschine ausführbar sind, die mit einem Ausführungsstack und Registern von durch diese Anweisungen manipulierten Operanden ausgestattet ist und es erlaubt, diesen Objektcode zu interpretieren. Das mitgeführte System ist an ein Lesegerät angeschlossen.

[0030] Bemerkenswert ist, dass nach der Erfassung eines Fernladebefehls und der Speicherung des das Programmfragment bildenden Objektcodes in dem überschreibbaren Speicher es darin beruht, für jedes Unterprogramm einen Schritt der Initialisierung des Typenstacks und der Tabelle der Registertypen mit Daten durchzuführen, die den Zustand der virtuellen Maschine zu Beginn der Ausführung des zeitweilig gespeicherten Objektcodes darstellen, eine Überprüfung des zeitweilig gespeicherten Objektcodes, Anweisung für Anweisung, durch Diskriminieren des Vorhandenseins, für jeden aktuellen Befehl, eines Ziels einer Verzweigungsanwei-

sung, eines Zieles eines Ausnahmeverwaltungsaufrufes oder eines Ziels eines Subroutinenaufrufs durchzuführen, und eine Aktualisierung der Wirkung der laufenden Anweisung auf die Datentypen des Typenstacks und die Tabelle der Registertypen in Abhängigkeit vom Vorhandensein eines Verzweigungsanweisungsziels, eines Subroutinenaufrufziels oder eines Ziels eines Ausnahmeverwaltungsaufrufs, durchzuführen. Die Überprüfung ist gelungen, wenn die Tabelle der Registertypen im Laufe einer Überprüfung aller Anweisungen nicht verändert wird, wobei der Überprüfungsprozess in Abwesenheit einer Veränderung Anweisung für Anweisung fortgesetzt wird, bis die Tabelle der Registertypen stabil ist. Andernfalls wird der Überprüfungsprozess unterbrochen.

[0031] Das erfindungsgegenständliche Verfahren zur Umwandlung eines Objektcodes eines Programmfragments in einen normierten Objektcode für das gleiche Programmfragment ist anwendbar auf einen Objektcode eines Programmfragments, in welchem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen zeigt, und für jede Verzweigungsanweisung der Stack-Variablentyp in Höhe dieser Verzweigung der gleiche ist wie in Höhe der Ziele dieser Verzweigung. Der erhaltene normierte Objektcode ist derart, dass die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen zeigt und der Ausführungsstack bei jeder Verzweigungszielanweisung leer ist.

[0032] Bemerkenswert ist, dass es darin beruht, für die Menge der Anweisungen des Objektcodes jede laufende Anweisung mit dem Datentyp des Ausführungsstacks vor und nach Ausführung dieser Anweisung zu kommentieren, wobei die Kommentierungsdaten mit Hilfe einer Analyse des auf diese Anweisung bezogenen Datenstroms berechnet werden, in den Anweisungen und jeder laufenden Anweisung das Vorhandensein von Verzweigungen, für die der Ausführungsstack nicht leer ist, zu erfassen, wobei die Erfassungsoperation ausgehend von den jeder laufenden Anweisung zugeordneten Stack-Variablentyp-Kommentierungsdaten durchgeführt wird. Ferner beruht es darin, bei Vorliegen eines nicht leeren Ausführungsstacks, Stack-Variablentransfer-Anweisungen jeweils beiderseits dieser Verzweigungen oder dieser Verzweigungsziele einzufügen, um den Inhalt des Ausführungsstacks vor der Verzweigung in zeitweilige Register zu entleeren und den Ausführungsstack anhand dieser zeitweiligen Register nach der Verzweigung wieder herzustellen, und anderenfalls keine Transferanweisung einzufügen. Durch dieses Verfahren kann so ein normierter Objektcode für das gleiche Programmfragment erhalten werden, bei dem der Ausführungsstack bei jeder Verzweigungsanweisung und Verzweigungszielanweisung leer ist, ohne dass die Ausführung des Programmfragments verändert wird.

[0033] Das Verfahren zur Umwandlung eines Objektcodes eines Programmfragments in einen normierten Objektcode des gleichen Programmfragments gemäß der vorliegenden Erfindung ist ferner anwendbar auf einen Objektcode eines Programmfragments, in dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören und ein von einer Anweisung dieses Objektcodes in ein Register geschriebener Operand von gegebenem Typ aus diesem gleichen Register von einer anderen Anweisung dieses Objektcodes mit dem gleichen gegebenen Datentyp wieder gelesen wird. Der erhaltene normierte Objektcode ist derart, dass die Operanden zu den von dieser Anweisung manipulierten Datentypen gehören, wobei in dem gesamten normierten Objektcode ein einziger und gleicher Datentyp einem gleichen Register zugewiesen wird.

[0034] Bemerkenswert ist, dass es darin beruht, für die Gesamtheit der Anweisungen des Objektcodes jede laufende Anweisung mit dem Datentyp der Register vor und nach Ausführung dieser Anweisung zu kommentieren, wobei die Kommentierungsdaten mit Hilfe einer Analyse des diese Anweisung betreffenden Datenstroms berechnet werden, und eine Neuzuteilung von mit diesen verschiedenen Typen verwendeten Ursprungsregistern durch Aufteilen dieser Ursprungsregister in verschiedene normierte Register vorzunehmen. Ein normiertes Register wird jedem verwendeten Datentyp zugeteilt. Eine Reaktualisierung der Anweisungen, die die auf diese normierten Register zugreifenden Operanden manipulieren, wird durchgeführt.

[0035] Das Protokoll zur Verwaltung eines Programmfragments, das Verfahren zur Überprüfung eines Programmfragments und die Verfahren zur Umwandlung von Objektcode von Programmfragmenten in normierten Objektcode und die entsprechenden Systeme gemäß der vorliegenden Erfindung werden eingesetzt bei der Entwicklung von umprogrammierbaren mitgeführten Systemen wie etwa Mikroprozessorkarten, insbesondere in Java-Umgebung.

[0036] Sie sind besser zu verstehen anhand der Lektüre der Beschreibung und der Betrachtung der beigefügten Zeichnungen, bei denen, neben den den Stand der Technik betreffenden **Fig.** 1a und 1b:

[0037] **Fig.** 2 ein Flussdiagramm darstellt, welches das Protokoll zur Verwaltung eines ferngeladenen Programmfragments in einem umprogrammierbaren mitgeführten System veranschaulicht,

[0038] **Fig.** 3a zur Verdeutlichung ein Flussdiagramm eines Verfahrens zur Überprüfung eines ferngeladenen Programmfragments gemäß dem Gegenstand der vorliegenden Erfindung zeigt,

[0039] **Fig.** 3b ein Diagramm darstellt, welches die Datentypen und Untertypisierungsbeziehungen verdeutlicht, die von dem Verwaltungsverfahren und dem Verfahren zur Überprüfung eines ferngeladenen Programmfragments gemäß der vorliegenden Erfindung eingesetzt werden,

[0040] **Fig.** 3c ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung eines Ver-

zweigungsbefehls darstellt,

[0041] **Fig.** 3d ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung einer Subroutinenaufrufanweisung, darstellt,

[0042] **Fig.** 3e ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung eines Ausnahmeverwaltungsziels, darstellt,

[0043] **Fig.** 3f ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung eines Ziels von inkompatiblen Verzweigungen, darstellt,

[0044] **Fig.** 3g ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung eines Fehlens eines Verzweigungsziels, darstellt,

[0045] **Fig.** 3h ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung der Wirkung der laufenden Anweisung auf den Typenstack, darstellt,

[0046] **Fig.** 3i ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung einer Registerleseanweisung, darstellt,

[0047] **Fig.** 3j ein Detail des Überprüfungsverfahrens gemäß **Fig.** 3a, betreffend die Verwaltung einer Registerschreibanweisung, darstellt,

[0048] **Fig.** 4a ein Flussdiagramm zur Verdeutlichung eines Verfahrens zur Umwandlung eines Objektcodes eines Programmfragments in einen normierten Objektcode für dieses Bleiche Programmfragment mit Verzweigungsanweisung bzw. Verzweigungszielanweisung mit leerem Stack, darstellt,

[0049] **Fig.** 4b ein Flussdiagramm zur Veranschaulichung eines Verfahrens zur Umwandlung eines Objektcodes eines Programmfragments in einen normierten Objektcode für dieses gleiche Programmfragment, der auf typisierte Register zugreift, darstellt, wobei jedem Register ein einziger spezifischer Datentyp zugewiesen ist,

[0050] **Fig.** 5a ein Detail der Ausführung des in **Fig.** 4a veranschaulichten Umwandlungsverfahrens darstellt,

[0051] **Fig.** 5b ein Detail der Ausführung des in **Fig.** 4b veranschaulichten Umwandlungsverfahrens darstellt,

[0052] **Fig.** 6 ein Funktionsschema der vollständigen Architektur eines Entwicklungssystems für ein normiertes Programmfragment und einer umprogrammierbaren Mikroprozessorkarte darstellt, die die Ausführung des Verwaltungsprotokolls und des Überprüfungsverfahrens eines Programmfragments gemäß dem Gegenstand der vorliegenden Erfindung ermöglicht.

[0053] Allgemein wird darauf hingewiesen, dass das Verwaltungsprotokoll, das Überprüfungsverfahren und das Verfahren zur Umwandlung eines ferngeladenen Programmfragments gemäß der vorliegenden Erfindung sowie die entsprechenden Systeme mit Hilfe einer Softwarearchitektur für das sichere Fernladen und Ausführen von Applets auf einem mitgeführten Informatiksystem mit geringen Ressourcen wie insbesondere Mikroprozessorkarten angewendet werden.

[0054] Allgemein wird darauf hingewiesen, dass die nachfolgende Beschreibung die Anwendung der Erfindung im Kontext von umprogrammierbaren Mikroprozessorkarten vom Typ JavaCard betrifft, siehe die zuvor in der Beschreibung erwähnte, bei der Fa. Sun Microsystems Inc. unter der Rubrik JavaCard Technology elektronisch verfügbare Dokumentation.

[0055] Die vorliegende Erfindung ist jedoch anwendbar auf jedes mitgeführte System, das über ein Fernladen eines Applets umprogrammierbar ist, das in dem Code einer virtuellen Maschine geschrieben ist, die einen Ausführungsstack, lokale Register oder Variablen umfasst und dessen Ausführungsmodell stark typisiert ist, wobei jede Anweisung des Codes des Applets nur für spezifische Datentypen anwendbar ist. Das Protokoll zur Verwaltung eines ferngeladenen Programmfragments auf einem umprogrammierbaren mitgeführten System gemäß der vorliegenden Erfindung wird nun in Verbindung mit in **Fig.** 2 detaillierter beschrieben.

[0056] In Verbindung mit der erwähnten Figur wird darauf hingewiesen, dass der Objektcode, der das Programmfragment oder Applet bildet, aus einer Folge von Anweisungen besteht, die von dem Mikroprozessor des mitgeführten Systems über die oben erwähnte virtuelle Maschine ausführbar sind. Die virtuelle Maschine erlaubt es, den erwähnten Objektcode zu interpretieren. Das mitgeführte System ist z. B. über eine serielle Verbindung an ein Endgerät angeschlossen.

[0057] Mit Bezug auf die erwähnte **Fig.** 2 beruht das erfindungsgegenständliche Verwaltungsprotokoll auf dem Niveau des mitgeführten Systems wenigstens darin, in einem Schritt **100a**, **100b** einen Befehl zum Fernladen dieses Programmfragments zu erfassen. So kann der Schritt **100a** in einem Schritt des Lesens des oben erwähnten Befehls und der Schritt **100b** in einem Schritt des Testens des gelesenen Befehls und der Überprüfung des Vorliegens eines Fernladebefehls beruhen.

[0058] Bei positiver Reaktion auf den oben erwähnten Schritt **100a**, **100b** der Erfassung eines Fernladebefehls beruht das erfindungsgegenständliche Protokoll anschließend darin, im Schritt **101** den Objektcode zu lesen, der das betreffende Programmfragment bildet, und den besagten Objektcode zeitweilig in dem Speicher des mitgeführten Informatiksystems zu speichern. Die besagte Operation des zeitweiligen Speicherns kann in dem überschreibbaren Speicher oder gegebenenfalls im Schreiblesespeicher des mitgeführten Systems durchgeführt werden, wenn letzterer eine ausreichende Kapazität aufweist. Der Schritt des Lesens des Objektcodes und des zeitweiligen Speicherns des letzteren in dem überschreibbaren Speicher wird in **Fig.** 2 als La-

den des Codes des Applets bezeichnet.

[0059] Auf den besagten Schritt folgt ein Schritt **102**, der darin beruht, die Gesamtheit des zeitweilig gespeicherten Objektcodes Anweisung für Anweisung einem Prozess der Überprüfung des besagten Objektcodes zu unterziehen.

[0060] Die Überprüfung beruht in wenigstens einem Schritt der Initialisierung des Typenstacks und der Tabelle der Datentypen, die den Zustand der virtuellen Maschine zu Beginn der Ausführung des zeitweilig gespeicherten Objektcodes darstellen, sowie in einer Folge von Überprüfungsschritten, Anweisung für Anweisung, durch Diskriminieren des Vorliegens, für jede mit I_i bezeichnete laufende Anweisung, eines Ziels wie etwa eines mit CIB bezeichneten Ziels einer Verzweigungsanweisung, eines Ziels eines Ausnahmeverwaltungsaufrufes oder eines Zieles eines Subroutinenaufrufs. Eine Überprüfung und eine Aktualisierung der Wirkung der laufenden Anweisung I_i auf den Typenstack und die Tabelle der Registertypen wird durchgeführt.

[0061] Wenn die Überprüfung in Schritt **103a** gelungen ist, beruht das erfindungsgegenständliche Protokoll darin, in Schritt **104** das ferngeladene Programmfragment in ein Verzeichnis von verfügbaren Programmfragmenten einzutragen und in Schritt **105** an das Lesegerät eine positive Empfangsbestätigung zu senden.

[0062] Umgekehrt beruht im Fall einer nicht gelungenen Überprüfung des Objektcodes in Schritt **103b** das erfindungsgegenständliche Protokoll darin, in einem Schritt **103c** jede Ausführung des zeitweilig gespeicherten Programmfragments auf dem mitgeführten System zu sperren. Der Schritt des Sperrens **103c** kann in unterschiedlicher Weise ausgeführt werden. Dieser Schritt kann, als nicht einschränkendes Beispiel, darin beruhen, in Schritt **106** das zeitweilig aufgezeichnete Programmfragment – bei Fehlen einer Aufzeichnung dieses Programmfragments im Verzeichnis von verfügbaren Programmfragmenten – zu löschen und in einem Schritt **107** an das Lesegerät einen Fehlercode zu senden. Die Schritte **107** und **105** können sequentiell nach den Schritten **106** bzw. **104** oder als Multitasking-Operation mit diesen ausgeführt werden.

[0063] Bezogen auf die gleiche **Fig.** 2 beruht bei negativer Reaktion auf den in der Erfassung eines Fernladebefehls beruhenden Schrittes **100b** das erfindungsgegenständliche Protokoll darin, in einem Schritt **108** einen Befehl zur Auswahl eines in einem Verzeichnis von Programmfragmenten verfügbaren Programmfragments zu erfassen und, bei positiver Reaktion auf den Schritt **108**, in welchem die Auswahl eines verfügbaren Programmfragments erfasst wird, in Schritt **109** dieses ausgewählte verfügbare Programmfragment aufzurufen, um es auszuführen. Auf den Schritt **109** folgt dann ein Schritt **110** des Ausführens des aufgerufenen verfügbaren Programmfragments mit Hilfe der virtuellen Maschine in Abwesenheit jeglicher dynamischer Überprüfung von Variablentypen, der Zugriffsrechte auf die von dem aufgerufenen verfügbaren Programmfragment manipulierten Objekte oder des Überlaufens des Ausführungsstacks bei der Ausführung jeder Anweisung.

[0064] In dem Fall, dass in Schritt **108** eine negative Antwort erhalten wird, wobei dieser Schritt darin beruht, einen Befehl zur Auswahl eines aufgerufenen verfügbaren Programmfragments zu erfassen, beruht das erfindungsgegenständliche Protokoll darin, in einem Schritt **111** die Verarbeitung der Standardbefehle des mitgeführten Systems vorzunehmen.

[0065] Was das Fehlen einer dynamischen Überprüfung des Typs oder des Zugriffsrechtes auf Objekte beispielsweise vom Typ JavaCard betrifft, sei gesagt, dass dieses Fehlen einer Überprüfung die Sicherheit der Karte nicht beeinträchtigt, da der Code des Applets notwendigerweise die Überprüfung erfolgreich durchlaufen hat.

[0066] Genauer gesagt ist die Überprüfung des Codes, die gemäß dem erfindungsgegenständlichen Verfahren auf der Mikroprozessorkarte oder dem mitgeführten Informatiksystem durchgeführt wird, selektiver als die übliche Codeüberprüfung für die virtuelle Java-Maschine wie in dem weiter oben in der Beschreibung erwähnten Werk mit dem Titel „The Java Virtual Machine Specification" beschrieben.

[0067] Allerdings kann der gesamte im Sinne des herkömmlichen Java-Überprüfers korrekte Code der virtuellen Java-Maschine in einen äquivalenten Code umgewandelt werden, der in der Lage ist, die auf der Mikroprozessorkarte durchgeführte Codeüberprüfung erfolgreich zu durchlaufen.

[0068] Es ist zwar denkbar, Java-Codes, die die oben erwähnten Überprüfungskriterien im Rahmen der Anwendung des erfindungsgegenständlichen Protokolls erfüllen, direkt zu schreiben, doch ist ein bemerkenswerter Gegenstand dieses Protokolls auch die Anwendung eines automatischen Umwandlungsverfahrens von beliebigem Standard-Java-Code in einen normierten Code für das gleiche Programmfragment, der zwangsläufig den oben zitierten angewendeten Überprüfungskriterien genügt. Das Verfahren zur Umwandlung in normierten Code und das entsprechende System werden in der Beschreibung später detailliert behandelt.

[0069] Eine genauere Beschreibung des Verfahrens zur Überprüfung eines Programmfragments oder Applets gemäß dem Gegenstand der vorliegenden Erfindung wird nun in Verbindung mit **Fig.** 3a und den folgenden Figuren gegeben.

[0070] Allgemein ist das erfindungsgegenständliche Überprüfungsverfahren im Rahmen des oben in Verbindung mit **Fig.** 2 beschriebenen erfindungsgegenständlichen Protokolls zur Verwaltung eines Programmfragments oder unabhängig davon anwendbar, um jeden für notwendig befundenen Überprüfungsprozess zu gewährleisten.

[0071] Allgemein ist ein Programmfragment durch einen Objektcode gebildet, der wenigstens ein Unterpro-

gramm, allgemeiner als Methode bezeichnet, umfasst, und bildet eine Folge von von dem Mikroprozessor des mitgeführten Systems über die virtuelle Maschine ausführbaren Anweisungen.

[0072] Wie in **Fig.** 3a gezeigt, beruht das Überprüfungsverfahren für jedes Unterprogramm darin, einen Schritt **200** der Initialisierung des Typenstacks und der Tabelle der Registertypen der virtuellen Maschine mit Daten durchzuführen, die den Zustand dieser virtuellen Maschine zu Beginn der Ausführung des Objektcodes darstellen, der Gegenstand der Überprüfung ist. Dieser Objektcode kann, wie oben in Verbindung mit der Anwendung des erfindungsgegenständlichen Protokolls beschrieben, zeitweilig gespeichert werden.

[0073] Auf den erwähnten Schritt **200** folgt dann ein Schritt **200a**, der darin beruht, das Lesen der laufenden Anweisung I_i mit Index i auf die erste Anweisung des Objektcodes zu positionieren. Auf den Schritt **200a** folgt ein Schritt **201**, der darin beruht, eine Überprüfung des besagten Objektcodes Anweisung für Anweisung durchzuführen, indem für jede mit I_i bezeichnete laufende Anweisung die Existenz eines Verzweigungsanweisungsziels CIB, eines Ziels eines Ausnahmeverwaltungsaufrufes CEM oder eines Zieles eines Subroutinenaufrufs CSR diskriminiert wird.

[0074] Auf den Überprüfungsschritt **201** folgt ein Schritt **202** der Überprüfung und Aktualisierung der Wirkung der laufenden Anweisung I_i auf die Datentypen des Typenstacks und der Tabelle der Registertypen in Abhängigkeit von der Existenz, für die durch eine andere Anweisung anvisierten laufenden Anweisung, eines Ziels einer Verzweigungsanweisung CIB, eines Ziels eines Subroutinenaufrufs CSR oder eines Ziels eines Ausnahmeverwaltungsaufrufes CEM.

[0075] Auf den Schritt **202** für die laufende Anweisung I_i folgt ein Schritt **203** des Prüfens, ob die letzte Anweisung erreicht ist, bezeichnet mit

I_i = letzte Anweisung des Objektcodes? Bei negativer Antwort auf den Test **203** geht der Prozess über zur nachfolgenden Anweisung **204**, mit i = i + 1 bezeichnet, und kehrt zu Schritt **201** zurück.

[0076] Die besagte Überprüfung in Schritt **202** ist gelungen, wenn die Tabelle der Registertypen im Laufe einer Überprüfung aller den Objektcode bildenden Anweisungen I_i nicht verändert wird. Zu diesem Zweck ist ein Test **205** des Vorliegens eines stabilen Zustandes der Tabelle der Registertypen vorgesehen. Dieser Test ist bezeichnet:

$\exists$? Stabiler Zustand der Tabelle der Registertypen

[0077] Bei positiver Antwort auf den Test **205** ist die Überprüfung gelungen.

[0078] Im Fall, wohingegen kein Fehlen einer Veränderung festgestellt wird, wird der Überprüfungsprozess wiederholt und durch Rückkehr zum Schritt **200a** neu gestartet. Es lässt sich zeigen, dass das Ende des Prozesses garantiert nach maximal NrxH Iterationen erreicht wird, wobei Nr die Zahl der verwendeten Register bezeichnet und H eine von der Untertypisierungsbeziehung abhängige Konstante ist.

[0079] Verschiedene Angaben zu den Typen der im Laufe des oben in Verbindung mit **Fig.** 3a beschriebenen Überprüfungsprozesses manipulierten Variablen werden nun in Verbindung mit **Fig.** 3b gemacht.

[0080] Die oben erwähnten Variablentypen umfassen wenigstens Klassenidentifikatoren, die den in dem zu überprüfenden Programmfragment definierten Klassen von Objekten entsprechen, numerische Variablentypen, die wenigstens einen Typ short, auf p bits ganzzahlig codiert, wobei p den Wert p = 16 annehmen kann, und einen Typ einer Rückkehradresse einer Sprunganweisung JSR umfasst, wobei dieser Adressentyp mit retaddr bezeichnet ist, einen Typ null der sich auf Verweise auf ungültige Objekte bezieht, einen Typ object, der sich auf die eigentliche Objekte bezieht, einen spezifischen Typ $\perp$, der die Schnittmenge aller Typen darstellt und den null-Wert Null entspricht, einen anderen spezifischen Typ T, der die Vereinigungsmenge aller Typen darstellt und jedem Typ von Werten entspricht. Bezogen auf **Fig.** 3b erfüllt die Menge der oben erwähnten Variablentypen eine Untertypisierungsbeziehung:

object $\in$ T;

short, retaddr $\in$ T;

$\perp \in$ null, short, retaddr

[0081] Ein spezifischeres Beispiel eines Überprüfungsprozesses wie in **Fig.** 3a dargestellt wird nun in Verbindung mit einem ersten Beispiel einer in der im Anhang beigefügten Tabelle T1 darstellten Datenstruktur gegeben.

[0082] Besagtes Beispiel betrifft ein in Java-Code geschriebenes Applet.

[0083] Der Überprüfungsprozess greift auf den Code des das zu überprüfende Applet bildenden Unterprogramms über einen Zeiger auf die zu überprüfende Anweisung I_i zu.

[0084] Der Überprüfungsprozess zeichnet die Größe und den Typ des Ausführungsstacks bei der laufenden Anweisung I_i auf, der in dem Beispiel der besagten Tabelle T1 saload entspricht.

[0085] Der Überprüfungsprozess zeichnet dann die Größe und den Typ des Ausführungsstacks bei der laufenden Anweisung im Typenstack über seinen Typenstackzeiger auf.

[0086] Wie zuvor in der Beschreibung erwähnt, spiegelt dieser Typenstack den Zustand des Ausführungsstacks der virtuellen Maschine bei der laufenden Anweisung I_i wieder. In dem in Tabelle T1 dargestellten Beispiel enthält bei der bevorstehenden Ausführung der Anweisung I_i der Stack drei Einträge: einen Verweis auf ein Objekt der Klasse C, einen Verweis auf eine Tabelle von auf p = 16 bits codierten ganzen Zahlen, den Typ

short[], und eine ganze Zahl von p = 16 Bits vom Typ short. Dies ist auch in dem Typenstack dargestellt, der ebenfalls drei Einträge enthält: C, den Typ der Objekte der Klasse C, short[], den Typ der Tabellen von ganzen Zahlen von p = 16 bits und short, den Typ der ganzen Zahlen mit p = 16 bits.

[0087] Eine andere bemerkenswerte Datenstruktur ist durch eine Tabelle von Registertypen gebildet, wobei diese Tabelle den Zustand der Register der virtuellen Maschine, d. h. der Register, die die lokalen Variablen speichern, wiederspiegelt.

[0088] Wieder bezogen auf das in Tabelle T1 angegebene Beispiel enthält der Eintrag 0 der Registertypentabelle den Typ C, d. h., dass bei der bevorstehenden Ausführung der laufenden Anweisung I_i = saload gewährleistet ist, dass das Register 0 einen Verweis auf ein Objekt der Klasse C enthält.

[0089] Die verschiedenen im Laufe der Überprüfung manipulierten und in der Registertypentabelle und dem Typenstack gespeicherten Typen sind in **Fig.** 3b dargestellt. Diese Typen umfassen:

– Klassenidentifikatoren CB, die den in dem Applet definierten spezifischen Klassen von Objekten entsprechen;

– die Grundtypen, wie etwa short, ganzzahlig auf p = 16 bits codiert, int1 und int2, die jeweils p signifikantesten bzw. am wenigstens signifikanten bits von auf z. B. 2p bits codierten ganzen Zahlen, oder retaddr der Rückkehradresse einer Anweisung, wie oben erwähnt;

– den Typ Null, der Verweise auf ungültige Objekte darstellt.

[0090] Was die Untertypisierungsbeziehung angeht, sei gesagt, dass ein Typ T1 Untertyp eines Typs T2 ist, wenn jeder gültige Wert des Typs T1 ebenfalls ein gültiger Wert des Typs T2 ist. Die Untertypisierung zwischen Klassenidentifikatoren spiegelt die Vererbungshierarchie zwischen Klassen des Applets wieder. Für die anderen Typen ist die Untertypisierung definiert durch das in **Fig.** 3b dargestellte Netzwerk, wobei $\perp$ Untertyp aller Typen ist und alle Typen Untertypisierung von T sind.

[0091] Der Ablauf des Prozesses zur Überprüfung eines ein Applet bildenden Unterprogramms ist, mit Bezug auf die oben erwähnte Tabelle T1, der folgende.

[0092] Der Überprüfungsprozess wird an jedem Unterprogramm des Applets unabhängig durchgeführt. Für jedes Unterprogramm führt der Prozess einen oder mehrere Überprüfungsdurchgänge an den Anweisungen des betreffenden Unterprogramms aus. Der Pseudocode des Überprüfungsprozesses ist in der beigefügten Tabelle T2 angegeben.

[0093] Der Prozess zur Überprüfung eines Unterprogramms beginnt mit der Initialisierung des Typenstacks und der Tabelle der Registertypen, wie in Tabelle T1 dargestellt, wobei diese Initialisierung den Zustand der virtuellen Maschine zu Beginn der Ausführung des untersuchten Unterprogramms darstellt.

[0094] Der Typenstack ist anfangs leer, der Stackzeiger ist gleich Null, und die Registertypen sind mit den Typen der Parameter des Unterprogramms initialisiert, was der Tatsache entspricht, dass die virtuelle Maschine die Parameter dieses Unterprogramms in diesen Registern übergibt. Die von dem Unterprogramm zugewiesenen Registertypen werden mit den Datentypen $\perp$ initialisiert, was die Tatsache darstellt, dass die virtuelle Maschine diese Register zu Beginn der Ausführung des Unterprogramms auf Null initialisiert.

[0095] Anschließend werden ein oder mehrere Überprüfungsdurchgänge an den Anweisungen und jeder laufenden Anweisung I_i des Unterprogramms durchgeführt.

[0096] Z. B. am Ende des durchgeführten Überprüfungsdurchgangs oder einer Folge von Durchgängen bestimmt der Überprüfungsprozess, ob die in der Tabelle der Registertypen enthaltenen, in der Tabelle T1 des Anhangs dargestellten Registertypen sich während des Überprüfungsdurchgangs geändert haben. Bei Fehlen einer Änderung ist die Überprüfung abgeschlossen, und ein Erfolgscode wird an das Hauptprogramm zurückgegeben, was es ermöglicht, in Schritt **105** des in **Fig.** 2 dargestellten Verwaltungsprotokolls die positive Empfangsbestätigung zu senden.

[0097] Bei Vorhandensein einer Veränderung der erwähnten Tabelle der Registertypen wiederholt der Überprüfungsprozess den Überprüfungsdurchgang, bis die in der Tabelle der Registertypen enthaltenen Registertypen stabil sind.

[0098] Der eigentliche Ablauf eines Überprüfungsdurchgangs, der ein oder mehrere Male bis zur Stabilität der Registertypen durchgeführt wird, wird nun in Verbindung mit den **Fig.** 3c bis 3j beschrieben.

[0099] Für jede laufende Anweisung I_i werden die folgenden Überprüfungen durchgeführt:

[0100] In Verbindung mit der **Fig.** 3a in Schritt **201** legt der Überprüfungsprozess fest, ob die laufende Anweisung I_i Ziel einer Verzweigungsanweisung, eines Subroutinenaufrufs oder einer Ausnahmeverwaltung ist, wie oben erwähnt. Diese Überprüfung erfolgt durch Untersuchen der im Code des Unterprogramms enthaltenen Verzweigungsbefehle und der diesem Unterprogramm zugeordneten Ausnahmeverwaltungen.

[0101] Wenn, mit Bezug auf die mit dem Schritt **201** beginnende **Fig.** 3c, die laufende Anweisung I_i Ziel einer Verzweigungsanweisung ist, wobei diese Bedingung durch einen mit I_i = CIB bezeichneten Test **300** überprüft wird, wobei die Verzweigung bedingt oder unbedingt ist, vergewissert sich der Überprüfungsprozess durch einen Test **301**, dass der Typenstack an diesem Punkt des Unterprogramms leer ist. Bei positiver Antwort auf den Test **301**, wird der Überprüfungsprozess mit einem Kontextfortsetzungsschritt, bezeichnet als Fortsetzung

A, fortgesetzt. Bei negativer Antwort auf den Test **301**, wenn der Typenstack nicht leer ist, scheitert die Überprüfung, und das Applet wird zurückgewiesen. Dieses Scheitern ist dargestellt durch den Schritt „Scheitern".

[0102] Bezogen auf die mit dem Schritt „Fortsetzung A" beginnende **Fig.** 3d überprüft, wenn die laufende Anweisung I_i das Ziel eines Subroutinenaufrufs ist, wobei diese Bedingung durch einen Test **304** I_i = CSR überprüft wird, der Überprüfungsprozess in einem Test **305**, dass die vorhergehende Anweisung I_{i-1} nicht sequentiell weitergeht. Diese Überprüfung wird durchgeführt durch einen Testschritt **305**, wenn die vorhergehende Anweisung eine unbedingte Verzweigung, eine Subroutinenrückkehr oder die Aufhebung einer Ausnahme darstellt. Der Test in Schritt **305** ist folgendermaßen bezeichnet:

$I_{i-1} = IB_{unbedingt}$, Rückkehr RSR oder Aufhebung L-EXCEPT.

[0103] Bei negativer Antwort auf den Test **305** schlägt der Überprüfungsprozess in einem Schritt „Scheitern" fehl. Bei positiver Antwort auf den Test **305** jedoch reinitialisiert der Überprüfungsprozess den Typenstack so, dass dieser genau einen Eintrag vom Typ retaddr einer Adresse für die Rückkehr von der oben erwähnten Subroutine enthält. Wenn die laufende Anweisung I_i in Schritt **304** nicht das Ziel eines Subroutinenaufrufs ist, wird der Überprüfungsprozess im Kontext in dem Schritt „Fortsetzung B" fortgesetzt.

[0104] Bezogen auf **Fig.** 3e wird, wenn die laufende Anweisung I_i Ziel einer Ausnahmeverwaltung ist, wobei diese Bedingung mit einem mit I_i = CEM bezeichneten Test **307** festgestellt wird, wobei CEM das Ziel einer Ausnahmeverwaltung darstellt, diese Bedingung über einen Test **307** festgestellt, der mit

I_i = CEM

bezeichnet ist.

[0105] Bei positiver Antwort auf den Test **307** überprüft der Prozess, dass die vorhergehende Anweisung einen unbedingten Sprungbefehl, eine Rückkehr von einer Subroutine oder die Aufhebung einer Ausnahme über einen Test **305**, bezeichnet mit

$I_{i-1} = IB_{unbedingt}$, Rückkehr RSR oder Aufhebung L-EXCEPT.

[0106] Bei positiver Antwort auf den Test **305** nimmt der Überprüfungsprozess eine Reaktualisierung des Typenstacks in einem Schritt **308** durch einen Eintrag von Ausnahmetypen, mit EXCEPT bezeichnet, vor, wobei auf den Schritt **308** ein Kontextfortsetzungsschritt C folgt. Bei negativer Reaktion auf den Test schlägt die Überprüfung mit dem „Scheitern" bezeichneten Schritt fehl. Das Programmfragment wird dann zurückgewiesen.

[0107] Wenn bezogen auf **Fig.** 3f die laufende Anweisung I_i das Ziel einer Mehrzahl von inkompatiblen Verzweigungen ist, wird diese Bedingung durch einen Test **309**, bezeichnet mit:

I_i = inkompatible XIB

festgestellt, wobei die inkompatiblen Verzweigungen z. B. eine unbedingte Verzweigung und ein Subroutinenaufruf oder auch zwei unterschiedliche Ausnahmeverwaltungen sind. Bei positiver Reaktion auf den Test **309**, wenn die Verzweigungen inkompatibel sind, scheitert der Überprüfungsprozess mit einem mit „Scheitern" bezeichneten Schritt, und das Programmfragment wird zurückgewiesen. Bei negativer Antwort auf den Test **309** wird der Überprüfungsprozess mit einem mit „Fortsetzung D" bezeichneten Schritt fortgesetzt. Der Test **309** wird eröffnet mit dem zuvor in der Beschreibung erwähnten Schritt „Fortsetzung C".

[0108] Wenn bezogen auf **Fig.** 3g die laufende Anweisung I_i nicht das Ziel einer Verzweigung ist, wobei diese Bedingung durch einen Test überprüft wird, der mit der oben erwähnten „Fortsetzung D" beginnt und bezeichnet ist mit

I_i ∃? Verzweigungsziele,

wobei ∃ das Existenzsymbol bezeichnet, geht der Überprüfungsprozess bei negativer Antwort auf den Test **310** weiter mit einem Übergang zu einer Aktualsierung des Typenstacks in einem Schritt **311**, wobei auf den Schritt **311** und auf die positive Antwort auf den Test **310** in Schritt **202** ein Kontextfortsetzungsschritt folgt, wie oben in der Beschreibung in Verbindung mit **Fig.** 3a beschrieben.

[0109] Eine detailliertere Beschreibung des Schritts der Überprüfung der Wirkung der laufenden Anweisung auf den Typenstack im oben beschriebenen Schritt **202** wird nun in Verbindung mit **Fig.** 3h geliefert.

[0110] Gemäß der erwähnten Figur kann dieser Schritt wenigstens einen Schritt **400** der Überprüfung umfassen, dass der Ausführungstypenstack wenigstens so viele Einträge enthält, wie die laufende Anweisung Operanden hat. Dieser Testschritt **400** ist bezeichnet mit:

Nbep ≥ NOpi,

wobei Nbep die Zahl der Einträge des Typenstacks und NOpi die Zahl von in der laufenden Anweisung enthaltenen Operanden bezeichnet.

[0111] Bei positiver Antwort auf den Test **300** folgt auf diesen Test ein Schritt **401** des Entstapelns des Typenstacks und der Überprüfung **401b**, dass die Typen der Einträge an der Spitze des Stacks Untertypen der Operandentypen der besagten laufenden Anweisung sind. In dem Testschritt **401a** sind die Operandentypen der Anweisung i mit TOpi und die Typen der Einträge an der Spitze des Stacks mit Targs bezeichnet.

[0112] In Schritt **401b** entspricht die Überprüfung einer Überprüfung der Untertypisierungsbeziehung "Targs Untertyp von TOpi".

[0113] Bei negativer Antwort auf den Test **400** und den Test **401b** scheitert der Überprüfungsprozess, was durch Verzweigen zum Schritt "Scheitern" dargestellt ist. Bei positiver Antwort auf den Test **401b** jedoch geht

der Überprüfungsprozess weiter und beruht darin, einen

– Schritt der Überprüfung der Existenz eines ausreichenden Speicherplatzes auf dem Typenstack zum Durchführen des Stapelns der Ergebnisse der laufenden Anweisung durchzuführen. Dieser Überprüfungsschritt wird durchgeführt mit einem Test **402**, bezeichnet mit

Stackplatz ≥ Ergebnisplatz

wobei jedes Element der Ungleichung den entsprechenden Speicherplatz bezeichnet.

[0114] Bei negativer Antwort auf den Test **402** scheitert der Überprüfungsprozess, was durch den Schritt „Scheitern" dargestellt ist. Bei positiver Antwort auf den Test **402** jedoch führt der Überprüfungsprozess die Stapelung der den Ergebnissen zugeordneten Datentypen in einem Schritt **403** durch, wobei die Stapelung auf dem diesen Ergebnissen zugeordneten Datentypenstack durchgeführt wird.

[0115] Als nicht einschränkendes Beispiel sei angegeben, dass für die Durchführung, nach **Fig.** 3h, der Überprüfung der Wirkung der laufenden Anweisung auf den Typenstack für eine laufende Anweisung, die durch eine Java-Anweisung saload gebildet ist, die dem Lesen eines auf p = 16 bits codierten ganzzahligen Elements in einer Tabelle von ganzen Zahlen entspricht, wobei diese Tabelle von ganzen Zahlen definiert ist durch die Tabelle von ganzen Zahlen und einem ganzzahligen Index in dieser Tabelle und das Ergebnis durch die an diesem Index in der Tabelle gelesene ganze Zahl, der Überprüfungsprozess sich vergewissert, dass der Typenstack wenigstens zwei Elemente enthält, dass die zwei Elemente an der Spitze des Typenstacks Untertypen von short[] bzw. short sind, den Entstapelungsprozess und anschließend den Stapelungsprozess mit dem Datentyp short als Typ des Ergebnisses vornimmt.

[0116] Außerdem beruht mit Bezug auf **Fig.** 3i für die Durchführung des Schritts der Überprüfung der Wirkung der laufenden Anweisung auf den Typenstack, wenn die laufende Anweisung I_i eine Anweisung, mit IR bezeichnet, zum Lesen eines Adressregisters n ist, wobei diese Bedingung durch einen mit $I_i = IR_n$ bezeichneten Test **404** überprüft wird, bei positiver Antwort auf den besagten Test **404** der Überprüfungsprozess darin, den Datentyp des Ergebnisses dieses Lesens in einem Schritt **405** durch Abfragen des Eintrags n der Tabelle der Registertypen zu überprüfen, dann die Wirkung der laufenden Anweisung I_i auf den Typenstack durch eine Operation **406a** des Entstapelns der Einträge des Stacks, die den Operanden dieser laufenden Anweisung entsprechen, und durch Stapeln **406b** des Datentyps dieses Ergebnisses zu bestimmen. Die Operanda der Anweisung I_i werden mit OP_i bezeichnet. Auf die Schritte **406a** und **406b** folgt eine Rückkehr zur Fortsetzung des Kontexts Fortsetzung F. Bei negativer Antwort auf den Test **404** wird der Überprüfungsprozess fortgesetzt durch die Fortsetzung des Kontexts Fortsetzung F.

[0117] Mit Bezug auf **Fig.** 3j besteht, wenn die laufende Anweisung I_i eine mit IW bezeichnete Anweisung zum Schreiben eines Adressregisters n ist, wobei diese Bedingung durch einen mit $I_i = IW_m$ bezeichneten Test überprüft wird, der Überprüfungsprozess darin, bei positiver Antwort auf den Test **407** in einem Schritt **408** die Wirkung der laufenden Anweisung auf den Typenstack und den Typ t des in das Adressregister n geschriebenen Operanden zu bestimmen, und dann, in einem Schritt **409**, den Eintrag des Typs in der Registertypentabelle an der Adresse n durch den Typ zu ersetzen, der unmittelbar höher als der zuvor gespeicherte Typ und der Typ t des in das Adressregister n geschriebenen Operanden ist. Ruf den Schritt **409** folgt eine Rückkehr zur Fortsetzung des Kontexts Fortsetzung **204**. Bei negativer Antwort auf den Test **407** wird der Überprüfungsprozess fortgesetzt mit einer Fortsetzung des Kontexts Fortsetzung **204**.

[0118] Wenn beispielsweise die laufende Anweisung I_i dem Schreiben eines Werts vom Typ D in ein Register mit Adresse 1 entspricht und der Typ des Registers 1 vor der Überprüfung der Anweisung C war, wird der Typ des Registers 1 ersetzt durch den Typ object, der in dem in **Fig.** 3b dargestellten Typennetz der niedrigste Typ ist, der höher ist als C und D.

[0119] Wenn z. B. die laufende Anweisung von I_i ein Lesen einer Anweisung aload-0 ist, die darin beruht, den Inhalt des Registers 0 zu stapeln, und wenn der Eintrag 0 der Tabelle der Registertypen C ist, stapelt der Überprüfer C auf dem Typenstack.

[0120] Ein Beispiel der Überprüfung eines in einer Java-Umgebung geschriebenen Unterprogramms wird nun in Verbindung mit den im Anhang beigefügten Tabellen T3 und T4 gegeben.

[0121] Die Tabelle T3 stellt einen spezifischen JavaCard-Code dar, der dem in dieser Tabelle enthaltenen Java-Unterprogramm entspricht.

[0122] Die Tabelle T4 veranschaulicht den Inhalt der Registertypentabelle und des Typenstacks vor der Überprüfung jeder Anweisung. Die Typenranforderungen für die Operanden der diversen Anweisungen sind alle eingehalten. Der Stack ist sowohl nach der Verzweigungsanweisung 5 zu der durch den Pfeil symbolisierten Anweisung 9, als auch vor dem besagten Verzweigungsziel 9 leer. Der Typ des Registers 1, der ursprünglich 1 war, wird null, die obere Schranke von null und ⊥, wenn die Anweisung 1 zum Speichern eines Werts vom Typ null im Register 1 untersucht wird, und wird dann zum Typ short[], der oberen Schranke des Typs short[] und des Typs null, wenn die Anweisung 8, das Speichern eines Werts vom Typ short[] im Register 1 verarbeitet wird. Da sich der Typ des Registers 1 während des ersten Überprüfungsdurchgangs geändert hat, wird ein

zweiter Durchgang durchgeführt, wobei die am Ende des ersten erhaltenen Registertypen verteilt werden. Dieser zweite Überprüfungsdurchgang gelingt wie der erste und verändert die Registertypen nicht. Der Überprüfungsprozess endet somit erfolgreich.

[0123] Verschiedene Bespiele für Fälle des Scheiterns des Überprüfungsprozesses werden nun anhand von vier Beispielen von inkorrektem Code in Verbindung mit der im Anhang beigefügten Tabelle T5 angegeben:

– Am Punkt a) der Tabelle T5 ist die Aufgabe des als Beispiels angegebenen Codes, zu versuchen, einen ungültigen Referenzobjektverweis zu erzeugen, indem ein arithmetischer Prozess an Zeigern verwendet wird. Er wird verworfen durch die Überprüfung der Typen der Argumente der Anweisung 2 sadd, welche verlangt, dass die zwei Argumente vom Typ short sind.

– An den Punkten b) und c) der Tabelle T5 ist die Aufgabe des Codes, zwei Versuche durchzuführen, eine beliebige ganze Zahl in einen Objektverweis umzuwandeln. Am Punkt b) wird das Register 0 gleichzeitig mit dem Typ short, Anweisung 0, und mit dem Typ null, Anweisung 5, verwendet. Folglich ordnet der Überprüfungsprozess dem Register 0 den Typ T zu und erfasst einen Typfehler, wenn das Register 0 als Ergebnis mit dem Typ object in Anweisung 7 zurückgegeben wird.

– Am Punkt c) der Tabelle T5 wird eine Anordnung von Verzweigungen vom Typ „if ... then ... else ..." verwendet, um an der Spitze des Stacks ein Ergebnis zu hinterlassen, das durch eine ganze Zahl oder durch einen Objektverweis gebildet ist. Der Überprüfungsprozess verwirft den Code, weil er erfasst, dass der Stack in Höhe der durch den Pfeil symbolisierten Verzweigung von der Anweisung 5 zur Anweisung 9 nicht leer ist.

– Schließlich enthält am Punkt d) der Tabelle T5 der Code eine Schleife, die bei jeder Iteration die Wirkung hat, dass eine weitere ganze Zahl an der Spitze des Stacks gestapelt wird, und die so nach einer bestimmten Zahl von Iterationen zu einem Stacküberlauf führt. Der Überprüfungsprozess verwirft diesen Code, indem er feststellt, dass der Stack in Höhe der Rückverzweigung von der Anweisung 8 zu Anweisung 0, symbolisiert durch den Rückwärtspfeil, nicht leer ist, d. h., dass der Stack an einem Verzweigungspunkt nicht leer ist.

[0124] Die verschiedenen oben in Verbindung mit den Tabellen T3, T4 und T5 angegebenen Beispiele zeigen, dass der Überprüfungsprozess, der den Gegenstand der vorliegenden Erfindung bildet, besonders effektiv ist und auf Applets und insbesondere auf Unterprogramme der letzteren anwendbar ist, für welche die die Typbedingungen des Stacks bzw. das Leersein des Typenstacks vor und bei Verzweigungsanweisungen oder Verzweigungszielen erfüllt sind.

[0125] Selbstverständlich impliziert ein solcher Prozess die Schreibung von diese Kriterien erfüllenden Objektcodes, wobei diese Objektcodes dem in der oben erwähnten Tabelle T3 eingeführten Unterprogramm entsprechen können.

[0126] Um die Überprüfung von Applets und von Unterprogrammen von existierenden Applets sicherzustellen, die nicht notwendigerweise den Überprüfungskriterien des erfindungsgegenständlichen Verfahrens genügen, insbesondere was die in Java-Umgebung geschriebenen Applets und Unterprogramme angeht, ist Ziel der Erfindung, Verfahren zur Umwandlung dieser Applets oder Unterprogramme in normierte Applets, die es erlauben, die Überprüfungstests des erfindungsgegenständlichen Überprüfungsverfahrens erfolgreich zu durchlaufen, und das ein solches Verfahren verwendende Verwaltungsprotokoll anzugeben.

[0127] Zu diesem Zweck ist Gegenstand der Erfindung die Anwendung eines Verfahrens und eines Programms zur Umwandlung eines herkömmlichen, ein Applet bildenden Objektcodes, wobei dieses Verfahren und dieses Umwandlungsprogramm außerhalb eines mitgeführten Systems oder einer Mikroprozessorkarte bei der Erzeugung des betreffenden Applets eingesetzt werden können.

[0128] Das Verfahren zur Umwandlung von Code in normierten Code nach der vorliegenden Erfindung wird nun ausschließlich zu Beispielzwecken im Rahmen der Java-Umgebung beschrieben.

[0129] Die von den existierenden Java-Compilern erzeugten JVM-Codes erfüllen unterschiedliche Kriterien, die nachfolgend aufgelistet sind:

C1: die Argumente jeder Anweisung gehören zu den von dieser Anweisung erwarteten Typen;

C2: der Stack läuft nicht über;

C'3: für jede Verzweigungsanweisung ist der Typ des Stacks in Höhe dieser Verzweigung derselbe wie in Höhe der für diese Verzweigung möglichen Ziele;

C'4: ein in einem Register an einem Punkt des Codes geschriebener Wert vom Typ t, der an einem anderen Punkt des Codes aus dem gleichen Register wieder gelesen wird, wird immer mit dem gleichen Typ t neu gelesen;

Die Ausführung des erfindungsgegenständlichen Überprüfungsverfahrens impliziert, dass die durch den Objektcode überprüften Kriterien C'3 und C'4 durch die nachfolgenden Kriterien C3 und
C4 ersetzt werden:

C3: der Stack ist bei jeder Verzweigungsanweisung und bei jedem Verzweigungsziel leer;

C4: ein gleiches Register wird im gesamten Code eines Unterprogramms mit einem einzigen Typ verwendet.

[0130] Mit Bezug auf die oben genannten Kriterien sei gesagt, dass die Java-Compiler nur die schwächeren Kriterien C'3 und C'4 garantieren, und dass der erfindungsgegenständliche Überprüfungsprozess und das entsprechende Überprüfungsprotokoll tatsächlich stärker einschränkende Kriterien C3 und C4 garantieren, mit denen die Ausführung und Verwaltung der Applets gewährleistet werden kann.

[0131] Der Begriff der Normierung, der die Umwandlung der Codes in normierte Codes abdeckt, kann unterschiedliche Aspekte aufweisen, insoweit die Ersetzung der Kriterien C'3 und C'4 durch die Kriterien C3 und C4 gemäß dem erfindungsgegenständlichen Überprüfungsprozess unabhängig durchgeführt werden kann, um zu gewährleisten, dass der Stack bei jeder Verzweigungsanweisung bzw. bei jedem Verzweigungsziel leer ist, dass die von dem Applet geöffneten Register typisiert sind, wobei jedem offenen Register ein einziger, für die Ausführung des betreffenden Applets zugewiesener Datentyp entspricht oder umgekehrt, um gemeinsam den gesamten erfindungsgegenständlichen Überprüfungsprozess zu erfüllen.

[0132] Das Verfahren zur Umwandlung eines Objektcodes in normierten Objektcode gemäß der Erfindung wird folglich mit zwei verschiedenen Ausführungsmodi beschrieben, einem ersten Ausführungsmodus, der der Umwandlung eines die Kriterien C1, C2, C'3, C'4 erfüllenden Objektcodes in einen normierten Objektcode entspricht, der die Kriterien C1, C2, C3, C'4 erfüllt, die einem auf eine leere Verzweigungsanweisung oder ein leeres Verzweigungsziel normierten Code entsprechen, dann, gemäß einer zweiten Ausgestaltung, bei der der herkömmliche Objektcode, der den gleichen Ausgangskriterien genügt, in einen normierten Objektcode umgewandelt wird, der z. B. die Kriterien C1, C2, C'3, C4 erfüllt, die einem typisierte Register verwendenden normierten Code entsprechen.

[0133] Die erste Ausgestaltung des erfindungsgegenständlichen Codeumwandlungsverfahrens wird nun in Verbindung mit **Fig.** 4a beschrieben. Bei der in **Fig.** 4a dargestellten Ausgestaltung soll der herkömmliche Ausgangscode den Kriterien C1 + C2 + C'3 entsprechen, und der durch die Umwandlung erhaltene normierte Code soll den Kriterien C1 + C2 + C3 entsprechen.

[0134] Gemäß der erwähnten Figur beruht der Umwandlungsprozess darin, für jede laufende Anweisung I_i des Codes oder des Unterprogramms jede Anweisung in einem Schritt **500** mit dem Datentyp des Stacks vor und nach Ausführung dieser Anweisung zu kommentieren. Die Kommentardaten sind mit AI_i bezeichnet und sind der betrachteten laufenden Anweisung durch die Beziehung $Ii \leftrightarrow AI_i$ zugeordnet. Die Kommentierungsdaten werden durch eine Analyse des diese Anweisung betreffenden Datenflusses berechnet. Die Datentypen vor und nach Ausführung der Anweisung sind mit tbe_1 bzw. tae_1 bezeichnet. Die Berechnung der Kommentierungsdaten durch Analyse des Datenflusses ist eine dem Fachmann vertraute herkömmliche Rechnung und wird deshalb nicht im Detail beschrieben.

[0135] Die im Schritt **500** durchgeführte Operation ist in der im Anhang eingeführten Tabelle T6 dargestellt, in welcher für ein Applet oder ein Applet-Unterprogramm mit zwölf Anweisungen die durch die Typen der Register und die Typen des Stacks gebildeten Kommentierungsdaten AI_i eingeführt sind.

[0136] Auf den besagten Schritt **500** folgt ein Schritt **500a**, der darin beruht, den Index i auf die erste Anweisung $I_i = I_1$ zu setzen. Auf den Schritt **500a** folgt ein Schritt **501**, der darin beruht, unter den Anweisungen und jeder laufenden Anweisung I_i die Existenz von mit IB bezeichneten Verzweigungen oder Verzweigungszielen CIB zu erfassen, für welche der Ausführungsstack nicht leer ist. Diese Erfassung **501** wird durchgeführt mit einem ausgehend von Kommentierungsdaten AI_i durchgeführten Test des jeder laufenden Anweisung zugewiesenen Typs von Stackvariablen, wobei der Test für die laufende Anweisung bezeichnet ist mit:
I_i ist IB oder CIB und Stack (AI) ≠ leer.

[0137] Bei positiver Antwort auf den Test **501**, d. h. bei Erfassung eines nicht leeren Ausführungsstacks, folgt auf den erwähnten Test ein Schritt, der darin beruht, Anweisungen zur Übertragung der Stackvariablen beiderseits dieser Verzweigungen IB oder dieser Verzweigungsziele CIB einzufügen, um den Inhalt des Ausführungsstacks vor der Verzweigung in temporäre Register zu entleeren und ausgehend von den temporären Registern nach der Verzweigung den Ausführungsstack wieder herzustellen. Der Einfügeschritt ist in **Fig.** 4a mit **502** bezeichnet. Auf ihn folgt ein Schritt **503** des Erreichens der letzten Anweisung, bezeichnet mit
I_i = letzte Anweisung?

[0138] Bei negativer Antwort auf den Test **503** wird eine Inkrementierung **504** i = i + 1 für den Übergang zu nächsten Anweisung und die Rückkehr zum Schritt **501** durchgeführt. Bei positiver Antwort auf den Test **503** wird ein Schritt Ende ausgeführt. Bei negativer Antwort auf den Test **501** wird das Umwandlungsverfahren fortgesetzt durch eine Verzweigung zum Schritt **503** in Abwesenheit der Einfügung einer Übertragungsanweisung. Mit der Ausführung des Verfahrens zur Umwandlung eines herkömmlichen Codes in einen normierten Code mit Verzweigungsanweisung bei leerem Stack, wie in **Fig.** 4a dargestellt, kann ein normierter Objektcode für das gleiche Ausgangsprogrammfragment erhalten werden, in welchem der Stack der Stackvariablen bei jeder Verzweigungsanweisung und jeder Verzweigungszielanweisung leer ist, ohne dass die Ausführung des Programmfragments verändert wird. Im Fall einer Java-Umgebung sind die Anweisung zur Übertragung von Daten zwischen Stack und Register load- und store-Anweisungen der virtuellen Java-Maschine.

[0139] Wiederum bezogen auf das Beispiel der Tabelle T6 erfasst das Umwandlungsverfahren ein Verzweigungsziel, wo der Stack nicht leer ist, bei der Anweisung 9. Es wird also vor der Verzweigungsanweisung 5,

die zu der besagten Anweisung 9 führt, eine Anweisung istore 1 eingefügt, um den Inhalt des Stacks in das Register 1 zu sichern und zu gewährleisten, dass der Stack bei der Verzweigung leer ist. Symmetrisch wird die Einfügung einer Anweisung iload 1 vor dem Anweisungsziel 9 durchgeführt, um den Inhalt des Stacks identisch zu seinem Zustand vor der Verzweigung wieder herzustellen. Schließlich wird eine Anweisung istore 1 nach der Anweisung 8 eingefügt, um das Gleichgewicht des Stacks auf den zwei Wegen zu garantieren, die zu der Anweisung **9** führen. Das Ergebnis der so durchgeführten Umwandlung in einen normierten Code ist in der Tabelle T7 dargestellt.

[0140] Die zweite Ausgestaltung des erfindungsgegenständlichen Umwandlungsverfahrens wird nun in Verbindung mit der **Fig.** 4b in dem Fall beschrieben, wo der herkömmliche Ausgangsobjektcode die Kriterien C1 + C'4 und der normierte Objektcode die Kriterien C1 + C4 erfüllt.

[0141] Bezogen auf die erwähnte **Fig.** 4b sei gesagt, dass das Verfahren bei dieser Ausgestaltung darin beruht, in einem Schritt **500**, der im Wesentlichen mit dem in **Fig.** 4a dargestellten identisch ist, jede laufende Anweisung I_i mit dem Datentyp der Register vor und nach Ausführung dieser Anweisung zu kommentieren. In gleicher Weise werden die Kommentierungsdaten AI_i mit Hilfe einer Analyse des Datenflusses betreffend dieser Anweisung berechnet.

[0142] Auf den Kommentierungsschritt **500** folgt ein Schritt, der darin beruht, eine Neuzuweisung der Register, einen mit **601** bezeichneten Schritt, durch Erfassen der mit unterschiedlichen Typen verwendeten Ausgangsregister, Aufteilen dieser Ausgangsregister in verschiedene normierte Register, wobei ein normiertes Register jedem verwendeten Datentyp zugewiesen ist, durchzuführen. Auf den Schritt **601** folgt ein Schritt **602** der Reaktualisierung der Anweisungen, welche die Operanden manipulieren, die auf die erwähnten normierten Register zugreifen. Auf den Schritt **602** folgt ein Schritt der Fortsetzung des Kontexts **302**.

[0143] Mit Bezug auf das in Tabelle T6 angegebene Beispiel sei gesagt, dass das Umwandlungsverfahren erfasst, dass das Register mit Rang 0, mit r0 bezeichnet, mit den zwei Typen object Anweisung 0 und 1 und int, Anweisung 9 und folgende, verwendet wird. Es wird dann eine Aufteilung des Ursprungsregisters r0 in zwei Register vorgenommen, das Register 0 für die Anwendung der object-Typen und das Register 1 für die Anwendungen vom Typ int. Die Verweise auf das Register 0 vom Typ int werden dann umgeschrieben durch Umwandeln in Verweise auf das Register 1, wobei der erhaltene normierte Code in der beigefügten Tabelle T8 angegeben ist.

[0144] Wie man sieht, wird in nichteinschränkender Weise in dem in Verbindung mit der erwähnten Tabelle T8 eingeführten Beispiel das neue Register 1 sowohl für die Normierung des Stacks als auch für die Erzeugung von typisierten Registern durch Aufteilung des Registers 0 in zwei Register verwendet.

[0145] Das Verfahren zur Umwandlung eines herkömmlichen Codes in einen normierten Code mit Verzweigungsanweisung bei leerem Stack, wie in **Fig.** 4a beschrieben, wird nun detaillierter in einer nichteinschränkenden, bevorzugten Ausgestaltung in Verbindung mit **Fig.** 5a beschrieben.

[0146] Diese Ausgestaltung betrifft den Schritt **501**, der darin beruht, in den Anweisungen und jeder laufenden Anweisung I_i das Vorhandensein einer Verzweigung IB bzw. eines Verzweigungsziels CIB zu erfassen, für die der Stack nicht leer ist.

[0147] Nach der Bestimmung der Zielanweisungen, wo der Stack nicht leer ist, wobei diese Bedingung in Schritt **504a** mit I_i Stack ≠ leer bezeichnet ist, beruht das Umwandlungsverfahren darin, in dem erwähnten Schritt **504a** diesen Anweisungen eine Menge von neuen Registern, eines pro in Höhe dieser Anweisungen aktivem Stackplatz zuzuordnen. Wenn somit i den Rang eines Verzweigungsziels bezeichnet, dessen zugeordneter Stacktyp nicht leer ist und vom Typ tpl_i bis tpn_i mit n > 0, Stack nicht leer, ist, teilt der Umwandlungsprozess n neue, noch nicht verwendete Register r_1 bis r_n zu und ordnet sie der entsprechenden Anweisung i zu. Diese Operation wird in Schritt **504a** vorgenommen.

[0148] Auf den Schritt **504a** folgt ein Schritt **504**, der darin beruht, jede erfasste Anweisung mit Rang i zu untersuchen und in einem Testschritt **504** das Vorhandensein eines Verzweigungsziels CIB oder einer Verzweigung IB zu diskriminieren. Der Schritt **504** ist in Form eines Test, bezeichnet mit:

$\exists$?CIB, IB und I_i = CIB

dargestellt.

[0149] In dem Fall, wo die Anweisung vom Rang i ein durch die vorhergehende Gleichung dargestelltes Verzweigungsziel CIB ist und der Stack der Stackvariablen in Höhe dieser Anweisung nicht leer ist, d. h. bei positiver Antwort auf den Test **504** für eine beliebige vorhergehende Anweisung vom Rang i – 1, die durch eine Verzweigung, eine Ausnahmeaufhebung oder eine Programmrückkehr gebildet ist, ist diese im Testschritt **505** realisierte Bedingung bezeichnet durch:

I_{i-1} = IB, Aufhebung EXCEPT, Programmrückkehr.

[0150] Die erfasste Anweisung von Rang i ist nur durch eine Verzweigung erreichbar. Bei positiver Antwort auf den erwähnten Test **505** beruht der Umwandlungsprozess darin, einen Schritt **506** durchzuführen, der darin beruht eine Menge von Ladeanweisungen vom Typ load ausgehend von der Menge von neuen Registern vor der betreffenden erfassten Anweisung mit Rang i einzufügen. Auf die Einfügeoperation **505** folgt eine Umlenkung **507** aller zu der erfassten Anweisung von Rang i führenden Verzweigungen auf die erste eingefügte La-

deanweisung load. Die Einfüge- und Umlenkungsanweisungen sind in der im Anhang beigefügten Tabelle T9 dargestellt.

[0151] Für jede sequentiell fortschreitende vorhergehende Anweisung vom Rang i – 1, d. h., wenn die laufende Anweisung vom Rang i sowohl durch eine Verzweigung als auch ausgehend von der vorhergehenden Anweisung erreichbar ist, wobei diese Bedingung durch den Test **508** überprüft wird und durch die Beziehungen:

$I_{i-1} \rightarrow I_i$

und

$IB \rightarrow I_i$

symbolisiert ist, beruht der Umwandlungsprozess in einem Schritt **509** des Einfügens einer Menge von Sicherungsanweisungen store in die Menge von neuen Registern vor der erfassten Anweisung vom Rang i und einer Menge von Ladeanweisungen load aus dieser Menge von neuen Registern. Auf den Schritt **509** folgt dann ein Schritt **510** des Umlenkens aller zu der erfassten Anweisung vom Rang i führenden Verzweigungen auf die erste eingefügte Ladeanweisung load.

[0152] In dem Fall, wo die erfasste Anweisung vom Rang i eine Verzweigung zu einer festgelegten Anweisung für jede erfasste Anweisung mit Rang i, gebildet durch eine unbedingte Verzweigung, ist, wobei diese Bedingung durch einen mit:

$I_i = Ib_{unbedingt}$

bezeichneten Test **511** überprüft wird, beruht der Umwandlungsprozess wie in **Fig.** 5a dargestellt darin, in einem Schritt **512** bei positiver Antwort auf den Test **511** vor der erfassten Anweisung mit Rang i eine Mehrzahl von Sicherungsanweisungen store einzufügen. Der Umwandlungsprozess fügt die n store-Anweisungen vor der Anweisung i ein, wie beispielhaft in Tabelle T11 dargestellt. Die store-Anweisungen adressieren die Register r_1 bis r_n, wobei n die Zahl der Register bezeichnet. Jedem neuen Register wird so die Sicherungsanweisung zugeordnet.

[0153] Für jede erfasste Anweisung mit Rang i, die durch eine unbedingte Verzweigung gebildet ist, und für eine Zahl mOp von mehr als 0 von dieser Verzweigungsanweisung manipulierten Operanden, wobei diese Bedingung in dem mit:

$I_i = IB_{bedingt}$

mit $mOp > 0$

bezeichneten Test **513** überprüft wird, beruht der Umwandlungsprozess bei positiver Antwort auf den erwähnten Test **513** darin, vor dieser erfassten Anweisung mit Rang i in einem Schritt **514** eine mit swap_x bezeichnete Anweisung zum Permutieren, an der Spitze der Stackvariablen, der mOp Operanden der erfassten Anweisung mit Rang i und der n nachfolgenden Werte durchzuführen. Durch diese Permutationsoperation kann die Menge der n der in der Menge von neuen Registern r_1 bis r_n zu sichernden Werte an die Spitze des Stacks der Stackvariablen gebracht werden. Auf den Schritt **514** folgt ein Schritt **515**, der darin beruht, vor der Anweisung mit Rang i eine Menge von Operationen store zum Sichern in die Menge der neuen Register r_1 bis r_n einzufügen. Auf den besagten Schritt **515** des Einfügens folgt seinerseits ein Schritt **516** des Einfügens, hinter die erfasste Anweisung mit Rang i, einer Menge von Anweisungen load zum Laden aus der Menge der neuen Register r_1 bis r_n. Die Menge der entsprechenden Einfügeoperationen ist in der im Anhang eingefügten Tabelle T12 dargestellt.

[0154] Aus Gründen der Vollständigkeit und mit Bezug auf **Fig.** 5a sei angegeben, dass bei negativer Antwort auf den Test **504** die Fortsetzung des Umwandlungsprozesses realisiert wird durch einen Schritt der Fortsetzung des Kontexts Fortsetzung **503**, dass auf die negative Antwort auf die Tests **505**, **508**, **511** und **513** eine Fortsetzung des Umwandlungsprozesses über einen Kontextfortsetzungsschritt Fortsetzung **503** folgt, und dass das Gleiche für die Fortsetzung der Operationen nach den erwähnten Umlenkungsschritten **507** und **510** bzw. Einfügeschritten **512** und **516** gilt.

[0155] Eine detailliertere Beschreibung des Normierungs- und Umwandlungsverfahrens eines Objektcodes in einen normierten Objektcode unter Verwendung von typisierten Registern, wie in **Fig.** 4b beschrieben, wird nun in Verbindung mit **Fig.** 5b gegeben. Diese Ausgestaltung betrifft insbesondere eine bevorzugte, nichteinschränkende Ausgestaltung des Schritts **601** der Neuzuweisung der Register durch Erfassen von mit unterschiedlichen Typen verwendeten Ursprungsregistern.

[0156] Mit Bezug auf die erwähnte **Fig.** 5b sei gesagt, dass der erwähnte Schritt **601** darin beruht, in einem Schritt **603** Lebensdauerintervalle, mit ID_j bezeichnet, jedes Registers r_j zu bestimmen. Diese Lebensdauerintervalle, englisch als „live range" oder „webs" bezeichnet, sind für ein Register r definiert als maximale Menge an partiellen Spuren, so dass das Register r an allen Punkten dieser Spuren lebendig (vivant) ist. Für eine detaillierte Definition dieser Begriffe kann auf das Werk mit dem Titel „Advanced Compiler Design and Implementation" herausgegeben von Steven S. Muchnik, Sektion 16.3, Morgan Kaufmann, 1997, zurückgegriffen werden. Der Schritt **603** ist bezeichnet durch die Beziehung:

$ID_j \leftrightarrow r_j$,

der zufolge jedem Register r_j ein entsprechendes Lebensdauerintervall ID_j zugeordnet wird.

[0157] Auf den erwähnten Schritt **603** folgt ein Schritt **604**, der darin beruht, in Schritt **604** den mit tp_j bezeich-

neten Hauptdatentyp jedes Lebensdauerintervalls ID_j zu bestimmen. Der Hauptdatentyp eines Lebensdauerintervalles ID_j für ein Register r_j ist definiert durch die obere Schranke der in diesem Register r_j durch die Sicherungsanweisungen store, die zu dem besagten Lebensdauerintervall gehören, gespeicherten Datentypen.

[0158] Auf den Schritt **604** folgt wiederum ein Schritt **605**, der darin beruht, einen Graphen von Interferenzen zwischen den zuvor in den Schritten **603** und **604** definierten Lebensdauerintervallen zu erzeugen, wobei dieser Interferenzgraph ein nicht orientierter Graph ist, von dem jeder Knoten durch ein Lebensdauerintervall gebildet ist und dessen Kanten, mit $a_{j1,j2}$ in **Fig.** 5b bezeichnet, zwischen zwei Knoten ID_j und ID_{j2} existieren, wenn ein Knoten eine an das Register des anderen Knotens adressierte Sicherungsanweisung enthält, oder umgekehrt. In **Fig.** 5b ist die Konstruktion des Interferenzgraphen symbolisch dargestellt, wobei diese Konstruktion anhand von dem Fachmann bekannten Rechentechniken durchgeführt werden kann. Für eine detailliertere Beschreibung der Konstruktion dieses Typs von Graphen kann man nützlicherweise auf das von Alfred V. Aho, Ravi Sethi und Jeffrey D. Ullman veröffentlichte Werk mit dem Titel „Compilers: Principles, Techniques and Tools", Addison-Wesley 1986, Sektion 9.7, zurückgreifen.

[0159] Nach dem Schritt **605** beruht das Normierungsverfahren wie in **Fig.** 5b dargestellt darin, eine Einheitlichkeit eines jedem Register r_j zugewiesenen Datentyps im Interferenzgraphen herzustellen, indem Kanten zwischen allen Paaren von Knoten des Interferenzgraphen eingefügt werden, solange zwei Knoten eines Paares von Knoten nicht den gleichen zugeordneten Hauptdatentyp haben. Es versteht sich, dass die Übersetzung des Einzigkeitscharakters eines jedem Register zugewiesenen Datentyps selbstverständlich der Übersetzung und Berücksichtigung des zuvor in der Beschreibung erwähnten Kriteriums C4 in dem Interferenzgraph entspricht. Auf den Schritt **606** folgt dann ein Schritt **607**, in welchem eine Instanziierung des Interferenzgraph durchgeführt wird, die allgemeiner als ein Schritt des Einfärbens (coloriage) des Interferenzgraphen bezeichnete Instantiierung nach den üblichen Techniken durchgeführt wird. Im Laufe des Schritts **607** ordnet der Umwandlungsprozess jedem Lebensdauerintervall ID_{jk} eine Registernummer rk so zu, dass zwei benachbarte Intervalle in dem Interferenzgraphen verschiedene Registernummern bekommen.

[0160] Diese Operation kann ausgehend von einem beliebigen angepassten Prozess durchgeführt werden. Als nichteinschränkendes Beispiel sei angegeben, dass ein bevorzugter Prozess darin beruhen kann

a) einen Knoten minimalen Grades in dem Interferenzgraphen zu wählen, wobei der minimale Grad definiert ist als eine minimale Zahl von benachbarten Knoten, und ihn aus dem Graphen zu entfernen. Dieser Schritt kann wiederholt werden, bis der Graph leer ist.

b) Jeder zuvor entfernte Knoten wird in den Interferenzgraphen in zu seiner Entfernung umgekehrter Reihenfolge wieder eingefügt, so dass der zuletzt entfernte der erste wieder eingeführte ist, usw., in entgegengesetzter Reihenfolge zur Reihenfolge der Entfernung. So kann jedem wieder eingefügten Knoten die kleinste Registernummer zugeordnet werden, die sich von den allen benachbarten Knoten zugeordneten Registernummern unterscheidet.

[0161] Schließlich schreibt in dem in **Fig.** 4b dargestellten Schritt **602** der Umwandlungs- und Neuverteilungsprozess die in dem Code des Unterprogramms des betreffenden Applets vorkommenden Registerzugriffsanweisungen neu. Ein Zugriff auf ein gegebenes Register in einem entsprechenden Lebensdauerintervall wird ersetzt durch einen Zugriff auf ein anderes Register, dessen Nummer während der auch als Einfärbungsphase bezeichneten Instanziierungsphase zugeordnet wurde.

[0162] Eine detailliertere Beschreibung eines mitgeführten Informatiksystems, welches die Anwendung des Verwaltungsprotokolls und des Prozesses der Überprüfung eines Programmfragments oder Applets gemäß dem Gegenstand der Erfindung erlaubt, und ein Entwicklungssystem für ein Applet werden nun in Verbindung mit **Fig.** 6 gegeben.

[0163] Zu dem das Bezugszeichen **10** tragenden mitgeführten System sei gesagt, dass dieses mitgeführte System vom umprogrammierbaren Typ ist und die wesentlichen Elemente wie in **Fig.** 1b dargestellt aufweist. Das erwähnte mitgeführte System wird als mit einem Endgerät durch eine serielle Verbindung verbunden angenommen, wobei das Endgerät seinerseits z. B. über ein lokales Netz, gegebenenfalls über ein entferntes Netz, mit einem Computer zur Entwicklung des Applets mit dem Bezugszeichen **20** verbunden ist. Auf dem mitgeführten System läuft ein Hauptprogramm, welches die auf der seriellen Verbindung durch das Endgerät gesendeten Befehle liest und ausführt. Außerdem können die Standardbefehle für eine Mikroprozessorkarte wie etwa die Standardbefehle des Protokolls ISO 7816, ausgeführt werden, wobei das Hauptprogramm ferner zwei zusätzliche Befehle erkennt, einen zum Fernladen eines Applets und den anderen zum Auswählen eines zuvor auf die Mikroprozessorkarte geladenen Applets.

[0164] Gemäß dem Gegenstand der vorliegenden Erfindung ist die Struktur des Hauptprogramms so realisiert, dass es wenigstens ein Programmmodul zur Verwaltung und Überprüfung eines ferngeladenen Programmfragments nach dem zuvor in der Beschreibung mit Bezug auf **Fig.** 2 beschriebenen Protokolls zur Verwaltung eines ferngeladenen Programmfragments aufweist.

[0165] Außerdem umfasst das Programmmodul auch ein Unterprogrammmodul zur Überprüfung eines ferngeladenen Programmfragments nach dem Überprüfungsverfahren wie zuvor in der Beschreibung in Verbin-

dung mit den **Fig.** 3a bis 3j beschrieben.

[0166] Zu diesem Zweck ist die Struktur der Speicher, insbesondere des nicht überschreibbaren permanenten Speichers bzw. ROM-Speichers so verändert, dass neben dem Hauptprogramm auch ein Modul **17** zur Protokollverwaltung und Überprüfung, wie zuvor erwähnt, vorhanden ist. Außerdem umfasst der nichtflüchtige überschreibbare Speicher vom EEPROM-Typ vorteilhafterweise ein Verzeichnis von Applets, mit **18** bezeichnet, welches die Ausführung des Verwaltungsprotokolls und des Überprüfungsprozesses gemäß der vorliegenden Erfindung ermöglicht.

[0167] Mit Bezug auf die gleiche **Fig.** 6 sei gesagt, dass das System zur Entwicklung des Applets gemäß dem Gegenstand der vorliegenden Erfindung, welches die Umwandlung eines herkömmlichen Objektcodes, wie oben in der Beschreibung erwähnt, der den Kriterien C1 + C2 + C'3 + C'4 genügt, im Rahmen der Java-Umgebung in einen normierten Objektcode für das gleiche Programmfragment ermöglicht, zugeordnet zu einem herkömmlichen Java-Compiler ein Codeumwandlungsmodul, mit **22** bezeichnet, umfasst, welches die Umwandlung des Codes in normierten Code gemäß der zuvor in der Beschreibung in Verbindung mit den **Fig.** 4a, 4b und 5a, 5b beschriebenen ersten und zweiten Ausgestaltung vornimmt. Die Normierung des ursprünglichen Objektcodes zu einem normierten Objektcode mit Verzweigungsanweisung bei leerem Stack und einen normierten Objektcode, der typisierte Register verwendet, wie oben in Beschreibung erwähnt, erlaubt es, die von dem erfindungsgegenständlichen Überprüfungsverfahren vorgegeben Überprüfungskriterien C3 und C4 zu erfüllen.

[0168] Auf das Codeumwandlungsmodul **22** folgt ein JavaCard-Wandler **23**, der die Übertragung über ein entferntes oder lokales Netz an das Endgerät und über die serielle Verbindung an die Mikroprozessorkarte **10** sicherzustellen erlaubt. Das in **Fig.** 6 dargestellte Applet-Entwicklungssystem **20** erlaubt es, die von dem Java-Compiler **21** ausgehend von den Java-Quellcodes des Applets erzeugten kompilierten Klassendateien in äquivalente Klassendateien umzuwandeln, die die von dem Verwaltungsprotokoll und dem Überprüfungsmodul **17**, die auf der Mikroprozessorkarte **10** mitgeführt sind, vorgegebenen zusätzlichen Anforderungen C3, C4 erfüllen. Diese umgewandelten Klassendateien werden auf der Karte durch den Standard-JavaCard-Wandler in ein fernladbares Applet konvertiert.

[0169] Diverse besonders bemerkenswerte Elemente aus der Menge der Elemente des Protokolls der Verfahren und der Systeme der vorliegenden Erfindung werden nun zur Erläuterung angegeben.

[0170] In Bezug auf die Überprüfungsprozesse des Standes der Technik, wie in der Einleitung der Beschreibung erwähnt, ist das erfindungsgegenständliche Überprüfungsverfahren insofern bemerkenswert, als es den Überprüfungsaufwand auf die Typisierungseigenschaften der Operanden konzentriert, die für die Sicherheit der Ausführung jedes Applets wesentlich sind, d. h. die Einhaltung der jeder Anweisung zugeordneten Typ-Bedingungen und das Fehlen eines Stacküberlaufs. Andere Überprüfungen erscheinen im Hinblick auf die Sicherheit nicht wesentlich, insbesondere die Überprüfung, dass der Code jedes Register korrekt initialisiert, bevor er es zum ersten Mal liest. Im Gegensatz dazu arbeitet das erfindungsgegenständliche Überprüfungsverfahren mit der Initialisierung aller Register auf Null ausgehend von der virtuellen Maschine bei der Initialisierung der Methode, um zu garantieren, dass das Lesen eines nicht initialisierten Registers die Sicherheit der Karte nicht kompromittieren kann.

[0171] Außerdem garantiert die von dem erfindungsgegenständlichen Überprüfungsverfahren vorgegebene Anforderung, dass bei jeder Verzweigungsanweisung oder jedem Verzweigungsziel der Stack leer sein soll, dass der Stack nach Ausführung der Verzweigung und vor Ausführung der Anweisung, zu der das Programm verzweigt hat, im gleichen Zustand, nämlich leer ist. Dieser Betriebsmodus garantiert, dass der Stack in einem kohärenten Zustand ist, egal, welcher Ausführungsweg durch den Code des Unterprogramms oder des betreffenden Applets verfolgt wird.

[0172] Die Kohärenz des Stacks ist so garantiert, auch bei Vorhandensein einer Verzweigung oder eines Verzweigungsziels. Im Gegensatz zu den herkömmlichen Verfahren und Systemen, bei denen es notwendig ist, den Typ des Stacks bei jedem Verzweigungsziel im Schreib-Lese-Speicher zu halten, was eine Menge an Schreiblesespeicher proportional zu Tp × Nb erfordert, dem Produkt aus der maximalen Größe verwendeten Ausführungsstacks und der Zahl der Verzweigungsziele im Code, benötigt das erfindungsgegenständliche Überprüfungsverfahren den Typ des Ausführungsstacks nur bei der gerade überprüften Anweisung, und es speichert nicht den Typ dieses Stacks an anderen Punkten des Codes. Folglich kommt das erfindungsgemäße Verfahren mit einer Menge an Schreiblesespeicher aus, die proportional zu Tp aber unabhängig von Nb und damit von der Länge des Codes des Unterprogramms oder Applets ist.

[0173] Die Anforderung des Kriteriums C4, demzufolge ein gegebenes Register mit einem einzigen Typ im gesamten Code eines Unterprogramms verwendet werden muss, garantiert, dass der erwähnte Code ein Register nicht inkohärent verwendet, z. B., indem er dort an einem Punkt des Programms eine ganze Zahl short schreibt und sie an einem anderen Punkt des Programms als einen Objektverweis liest.

[0174] Bei den im Stand der Technik beschriebenen Überprüfungsprozessen, insbesondere in der Java-Spezifikation mit dem Titel „The Java Virtual Machine Specification", herausgegeben von Tim Lindholm und Frank Yellin, bereits zitiert, ist es zum Garantieren der Kohärenz der erwähnten Verwendungen über die Verzwei-

gungsanweisungen hinweg notwendig, eine Kopie der Tabelle der Registertypen bei jedem Verzweigungsziel im Schreiblesespeicher zu halten. Diese Operation erfordert eine Menge an Schreiblesespeicher proportional zu $T_r \times N_b$, wobei T_r die Zahl der von dem Unterprogramm verwendeten Register und N_b die Zahl der Verzweigungsziele in dem Code des Unterprogramms ist.

[0175] Im Gegensatz dazu arbeitet der erfindungsgegenständliche Überprüfungsprozess mit einer globalen Tabelle von Registertypen und ohne Speicherung einer Kopie an verschiedenen Punkten des Codes im Schreiblesespeicher. Folglich ist der zur Durchführung des Überprüfungsprozesses erforderliche Schreiblesespeicher proportional zu T_r, aber unabhängig von N_b und damit von der Länge des Codes des betreffenden Unterprogramms.

[0176] Der Anforderung, der zufolge ein gegebenes Register mit dem gleichen Typ an allen Punkten, d. h. bei jeder Anweisung des betreffenden Codes, verwendet wird, vereinfacht wesentlich und signifikant die Überprüfung der Unterprogramme. Bei den herkömmlichen Überprüfungsprozessen und bei Fehlen einer solchen Anforderung muss der Überprüfungsprozess sicherstellen, dass die Unterprogramme eine strenge Stack-Disziplin einhalten und muss den Körper der Unterprogramme im Hinblick auf den Typ bestimmter Register in vielgestaltiger Weise überprüfen.

[0177] Zusammenfassend erlaubt es der erfindungsgegenständliche Überprüfungsprozess im Vergleich zu den herkömmlichen Techniken einerseits, die Größe des Programmcodes zu verringern, mit dem das Überprüfungsverfahren durchgeführt werden kann, und andererseits den Verbrauch an Schreiblesespeicher bei den Überprüfungsoperationen zu verringern, wobei der Komplexitätsgrad von der Form $O(T_p + P_r)$ im Fall des erfindungsgegenständlichen Überprüfungsprozesses ist, anstelle von $O(T_p + T_r) \times N_b)$ bei den herkömmlichen Überprüfungsprozessen, und wobei dennoch die gleichen Garantien im Hinblick auf die Sicherheit der Ausführung des überprüften Codes erreicht werden.

[0178] Schließlich wird der Prozess der Umwandlung eines herkömmlichen Ursprungscodes in einen normierten Code durch lokalisierte Umwandlung des Codes in Abwesenheit der Übertragung von Zusatzinformationen an das Überprüfungsorgan, d. h. an die Mikroprozessorkarte oder das mitgeführte Informatiksystem, durchgeführt.

[0179] Was das Verfahren zur Neuzuweisung von Registern, wie in **Fig.** 4b und 5b beschrieben, angeht, so unterscheidet sich dieses Verfahren von den insbesondere in dem Patent US 4,571,678 und dem Patent US 5,249,295 beschriebenen bekannten herkömmlichen Verfahren dadurch, dass

– die Registerneuzuweisung gewährleistet, dass ein gleiches Register nicht zwei Intervallen zugeteilt werden kann, die verschiedene Haupttypen besitzen, was garantiert, dass ein gegebenes Register in dem gesamten Code mit dem gleichen Typ verwendet wird; und

– dass die existierenden und in den oben zitierten Dokumenten beschriebenen Registerzuweisungsalgorithmen eine feste Zahl von Registern voraussetzen und versuchen, die im Englischen als „spills" bezeichneten Übertragungen zwischen Registern und Stack zu minimieren, wohingegen die Neuzuweisung der Register gemäß dem Gegenstand der vorliegenden Erfindung in einem Rahmen arbeitet, wo die Gesamtzahl der Register variabel ist, so dass es keine Veranlassung gibt, Übertragungen zwischen Registern und Stack durchzuführen, wenn ein Prozess zur Minimierung der Gesamtzahl von Registern eingesetzt wird.

[0180] Das Protokoll zur Verwaltung eines ferngeladenen Programmfragments auf einem mitgeführten System und die Verfahren zur Überprüfung dieses ferngeladenen Programmfragments bzw. zur Umwandlung dieses Objektcodes eines ferngeladenen Programmfragments gemäß der vorliegenden Erfindung können selbstverständlich softwaremäßig eingesetzt werden.

[0181] In diesem Zusammenhang betrifft die Erfindung gleichfalls ein direkt in den internen Speicher eines programmierbaren mitgeführten Systems ladbares Computerprogrammprodukt, wobei dieses mitgeführte System das Fernladen eines durch einen Objektcode, einer von dem Mikroprozessor des mitgeführten Systems ausführbaren Folge von Anweisungen, gebildeten Programmfragments über eine virtuelle Maschine erlaubt, die mit einem Ausführungsstack und Registern oder Variablen ausgestattet ist, die von diesen Anweisungen manipuliert werden, um die Interpretation dieses Objektcodes zu ermöglichen. Das entsprechende Computerprogrammprodukt umfasst Objektcodeabschnitte zur Ausführung des Protokolls zur Verwaltung eines ferngeladenen Programmfragments auf diesem mitgeführten System, wie in den oben in der Beschreibung beschriebenen **Fig.** 2 und 6 dargestellt, wenn das mitgeführte System an ein Terminal angeschlossen ist und dieses Programm vom Mikroprozessor des mitgeführten Systems über die virtuelle Maschine ausgeführt wird.

[0182] Die Erfindung betrifft auch ein direkt in den internen Speicher eines umprogrammierbaren mitgeführten Systems wie etwa einer mit einem überschreibbaren Speicher versehenen Mikroprozessorkarte ladbares Computerprogrammprodukt, wie in Verbindung mit **Fig.** 6 dargestellt. Dieses Computerprogrammprodukt umfasst Objektcodeabschnitte zur Ausführung der Schritte zur Überprüfung eines auf dieses mitgeführte System ferngeladenen Programmfragments, wie oben in der Beschreibung in Verbindung mit den **Fig.** 3a und 3j dargestellt und beschrieben. Diese Überprüfung wird ausgeführt, wenn dieses mitgeführte System an ein Endgerät angeschlossen ist und dieses Programm von dem Mikroprozessor dieses mitgeführten Systems über die

virtuelle Maschine ausgeführt wird.

[0183] Die Erfindung betrifft auch ein Computerprogrammprodukt, wobei dieses Computerprogrammprodukt Objektcodeabschnitte zur Ausführung der Schritte des Verfahrens zur Umwandlung des Objektcodes eines Programmfragments in einen normierten Objektcode für das gleiche Programmfragment umfasst, wie in den **Fig.** 4a, 4b bzw. 5a, 5b sowie in **Fig.** 6, wie zuvor in der Beschreibung beschrieben, umfasst.

[0184] Die vorliegende Erfindung betrifft auch ein Computerprogrammprodukt, das auf einem Träger aufgezeichnet ist, der in einem umprogrammierbaren mitgeführten System verwendbar ist, z. B. einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte, wobei dieses mitgeführte System das Fernladen eines durch einen von dem Mikroprozessor ausführbaren Objektcode gebildeten Programmfragments über eine virtuelle Maschine erlaubt, die mit einem Ausführungsstack und lokalen Variablen oder Registern ausgestattet ist, die von diesen Anwei sungen manipuliert werden, um die Interpretation des Objektcodes zu ermöglichen. Das besagte Computerprogrammprodukt umfasst wenigstens ein Modul von vom Mikroprozessor des mitgeführten Systems über die virtuelle Maschine lesbaren Programmen zum Steuern der Ausführung einer Verwaltungsprozedur zum Fernladen eines ferngeladenen Programmfragments, wie in **Fig.** 2 dargestellt und zuvor in der Beschreibung beschrieben, ein Modul von durch den Mikroprozessor über die virtuelle Maschine lesbaren Programmen zum Steuern der Ausführung einer Überprüfungsprozedur, Anweisung für Anweisung, des das Programmfragment bildenden Objektcodes, wie in Verbindung mit den **Fig.** 3a bis 3j in der vorhergehenden Beschreibung beschrieben und erläutert, und ein Modul von von dem Mikroprozessor dieses mitgeführten Systems über die virtuelle Maschine lesbaren Programmen zum Steuern der Ausführung eines ferngeladenen Programmfragments nach oder bei Fehlen einer Umwandlung des Objektcodes dieses Programmfragments in normierten Objektcode für das gleiche Programmfragment, wie in **Fig.** 2 dargestellt.

[0185] Das besagte Computerprogrammprodukt umfasst auch ein Modul von durch den Mikroprozessor über die virtuelle Maschine lesbaren Programmen zum Steuern der Sperrung der Ausführung, auf dem mitgeführten System, des Programmfragments im Fall einer nicht gelungenen Überprüfungsprozedur des besagten Programmfragments, wie zuvor in der Beschreibung in Verbindung mit **Fig.** 2 dargestellt und beschrieben.

ANHÄNGE

TABELLE 1

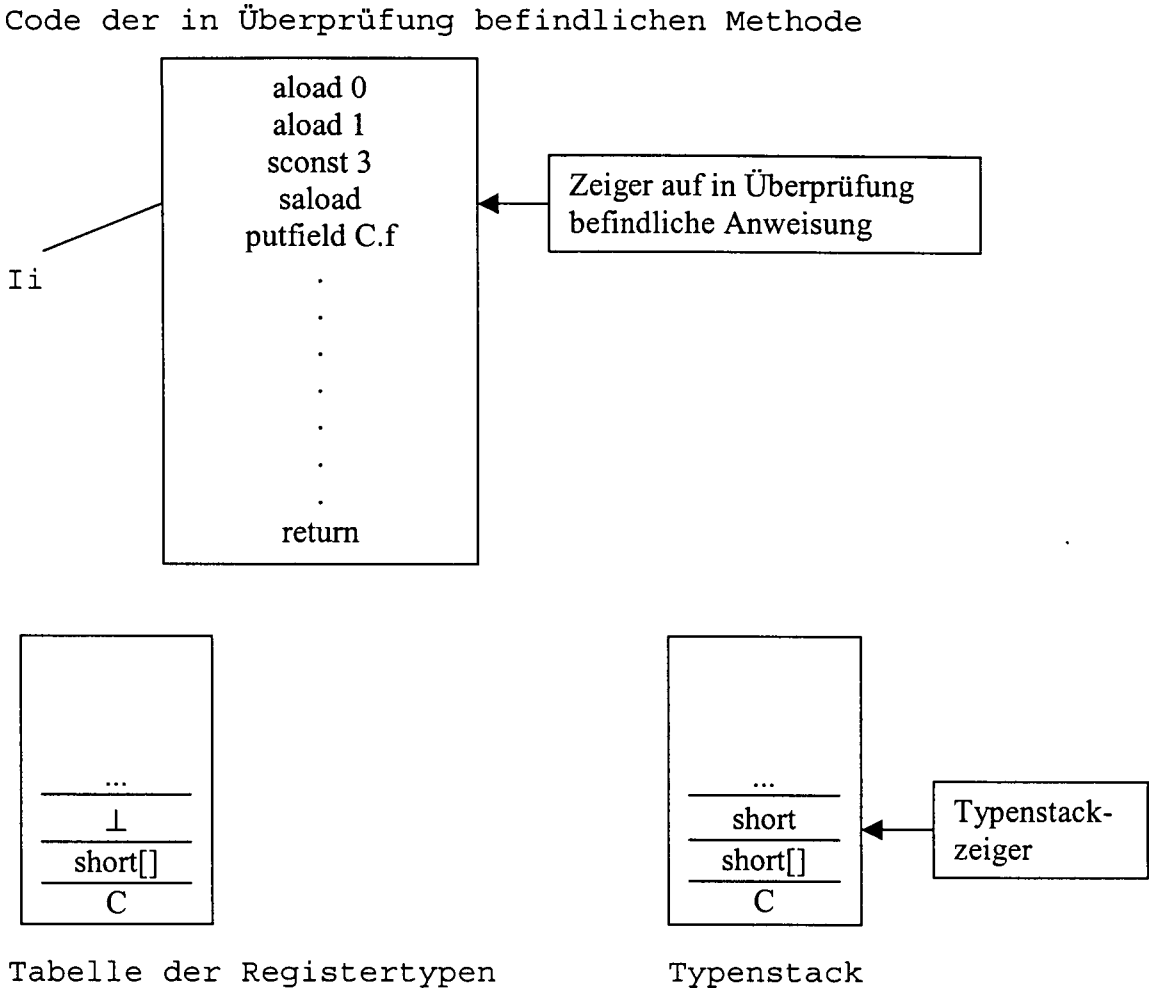


TABELLE 2

Pseudo-Code des Überprüfungsmoduls

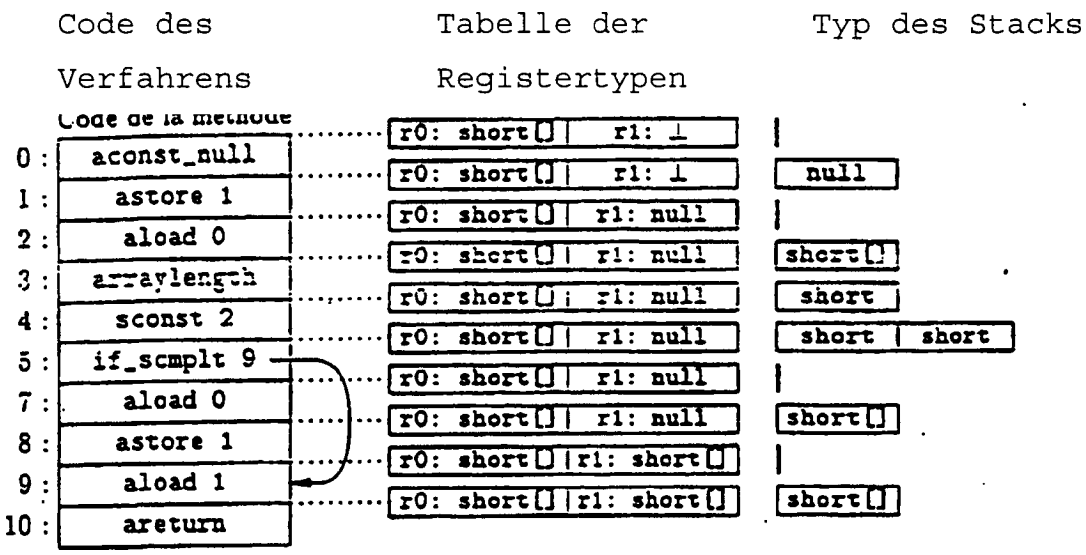
PSEUDO-CODE DES ÜBERPRÜFUNGSMODULS

[0186] Verwendete globale Variablen:
T_r von der laufenden Methode deklarierte Registerzahlen
T_P von der laufenden Methode deklarierte maximale Stackgröße
tr[T_r] Registertypentabelle (**402** in **Fig.** 4)
tp[T_p] Typenstack (**403** in **Fig.** 4)
pp Stackzeiger (**404** in **Fig.** 4)
chg Flag, welches angibt, ob tr verändert ist
Initialisieren pp ← 0
Initialisieren tp[0] ... tp[n – 1] anhand der Typen der n Argumente der Methode
Initialisieren von tp[n] ... tp[T_r – 1] auf ⊥
Initialisieren von chg auf wahr
Solange chg wahr ist:
chg auf falsch setzen
sich auf erste Anweisung der Methode positionieren solange Ende der Methode nicht erreicht ist:
wenn laufende Anweisung Ziel einer Verzweigungsanweisung ist:
Wenn pp ≠ 0, Scheitern der Überprüfung
wenn laufende Anweisung Ziel eines Subroutinenaufrufs ist:
Wenn vorhergehende Anweisung in Folge weitergeht, Scheitern
Setzen tp[0] ← reatddr und pp ← 1
wenn laufende Anweisung eine Ausnahmeverwaltung der Klasse C ist:
Wenn vorhergehende Anweisung in Folge weitergeht, Scheitern
Setzen tp[0] ← C und pp ← 1
wenn laufende Anweisung ein Ziel unterschiedlicher Arten ist:
Scheitern der Überprüfung
Bestimmen der Typen a_1, ..., an der Argumente der Anweisung
Wenn pp < n, Scheitern (Stacküberlauf) für i = 1, ..., n:
wenn tp[pp – n – i – 1] nicht Untertyp von a_i ist, Scheitern
Setzen pp ← pp – n
Bestimmen der Typen r_1, ..., r_m der Ergebnisse der Anweisung
wenn p + m ≥ t_p, Scheitern (Stacküberlauf)
für i = 1, ..., m, Setzen tp[pp ÷ i – 1] ← r_i
Setzen pp ← pp + m
wenn laufende Anweisung ein Schreiben in ein Register r ist:
Bestimmen des Typs t des in das Register geschriebenen Werts
Setzen tr[r] ← untere Schranke (t, tr[r])
Wenn tr[r] verändert ist, Setzen chg ← wahr
wenn laufende Anweisung eine Verzweigung ist:
Wenn pp ≠ 0, Scheitern der Überprüfung
weiter zur nächsten Anweisung
Zurückgeben eines Codes für erfolgreiche Überprüfung

TABELLE T3

```
Static short [] meth(short [] Tabelle)
{
     short [] Ergebnis = null
     if (Tabelle.Länge >= 2) Ergebnis = Tabelle;
     return Tabelle
}
```

TABELLE T4
Erste Iteration über den Code der Methode:



Zweite Iteration über den Code des Verfahrens:

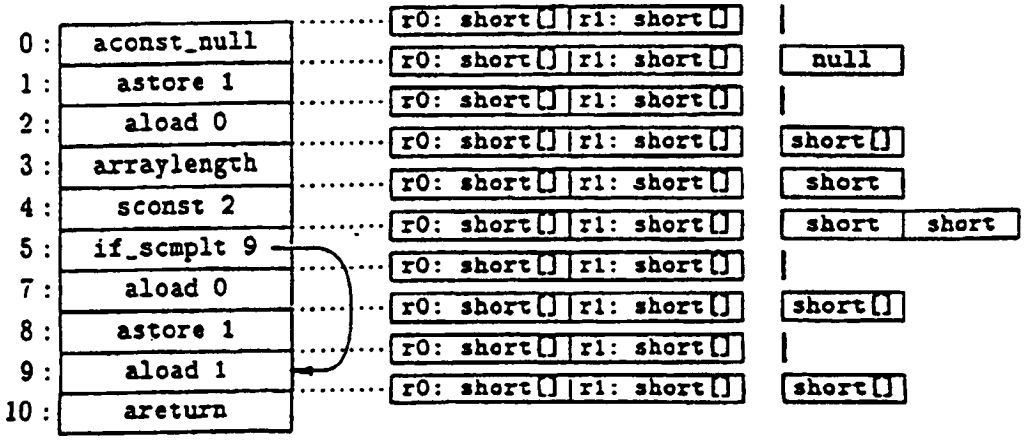
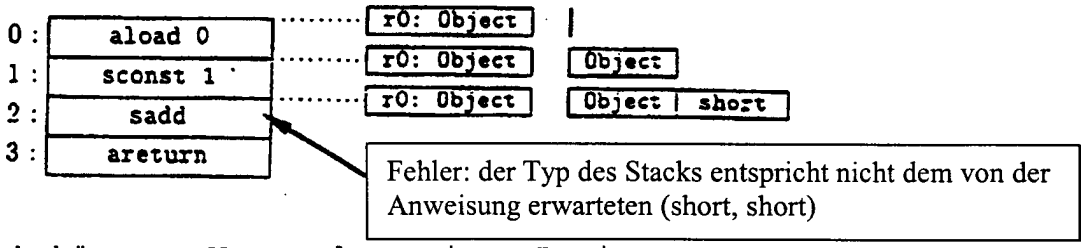
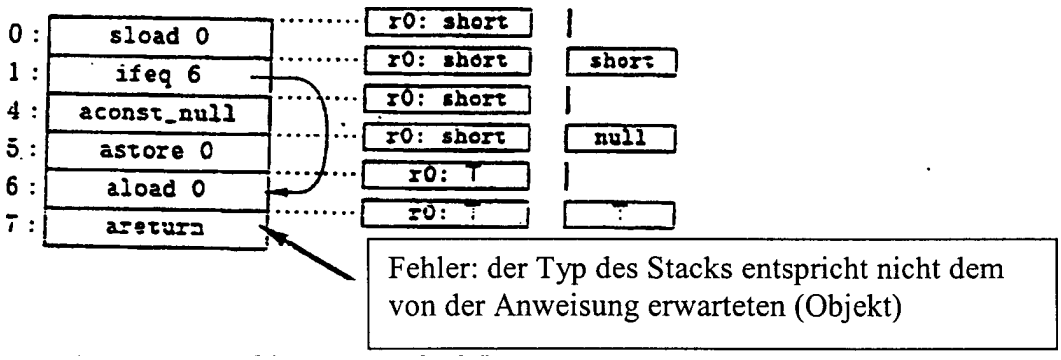


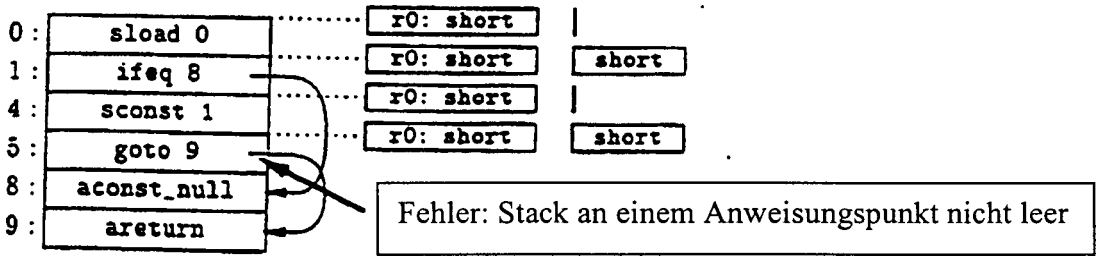
TABELLE T5
(a) Verletzung der Typanforderungen bei den Argumenten einer Anweisung:



Inkohärente Verwendung eines Registers:



(c) Verzweigungen, die zu Inkohärenzen am Stack führen:



(d) Stacküberlauf innerhalb einer Schleife:

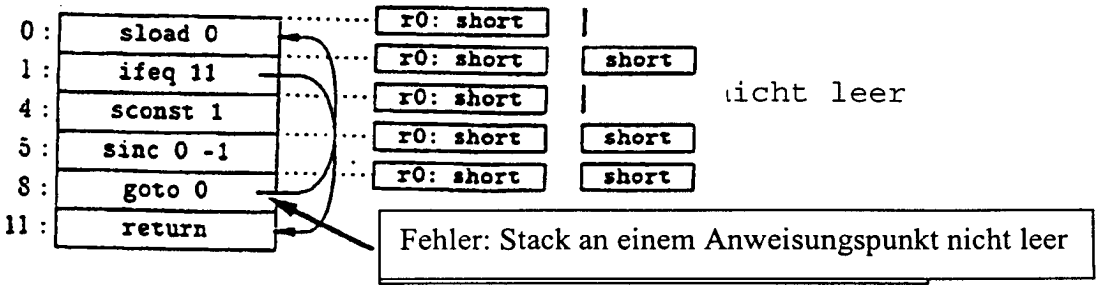


TABELLE T6

(a) Ursprünglicher Code der Methode, kommentiert mit den Typen der Register und des Stacks:

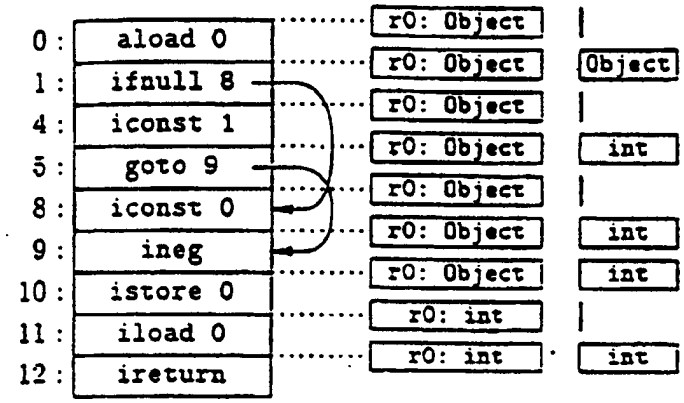


TABELLE T7

(b) Code des Verfahrens nach Normierung des Stacks in Höhe der Verzweigung 5 → 9:

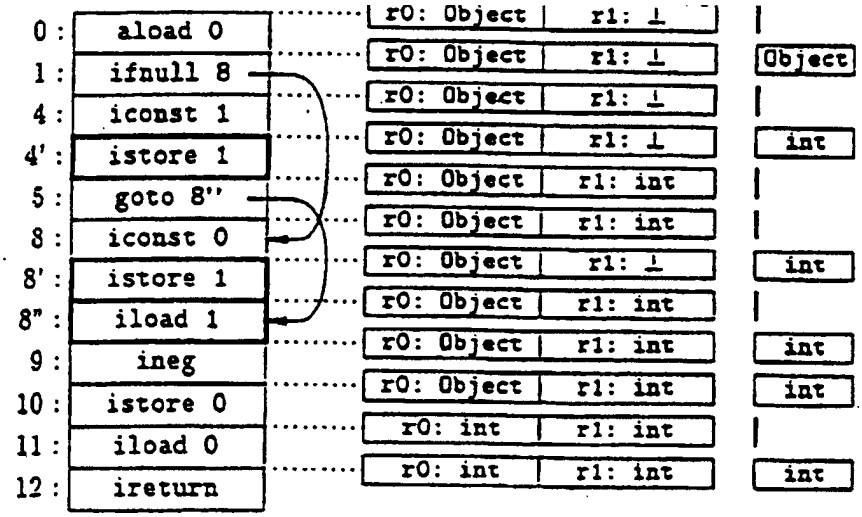


TABELLE T8

(c) Code des Verfahrens nach Neuzuweisung der Register

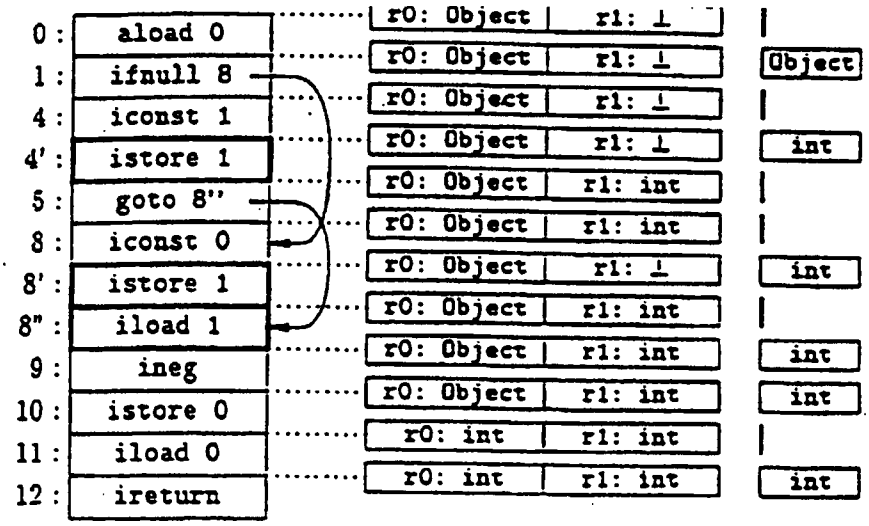


TABELLE T9

(a) Verzweigungsziel, vorhergehende Anweisung nicht in der

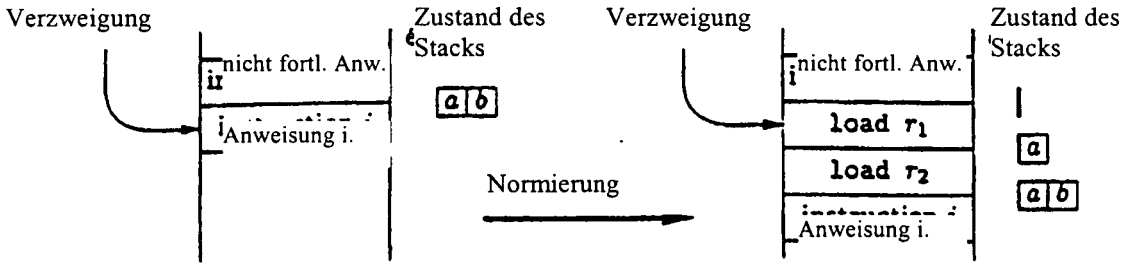


TABELLE T10
(b) Verzweigungsziel, vorhergehende Anweisung in Folge fortlaufend:

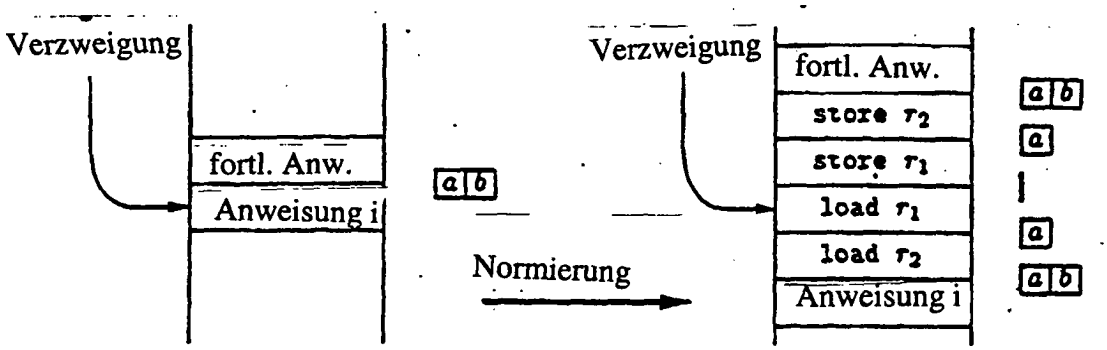


TABELLE T11
(c) Unbedingte Verzweigung ohne Argumente:

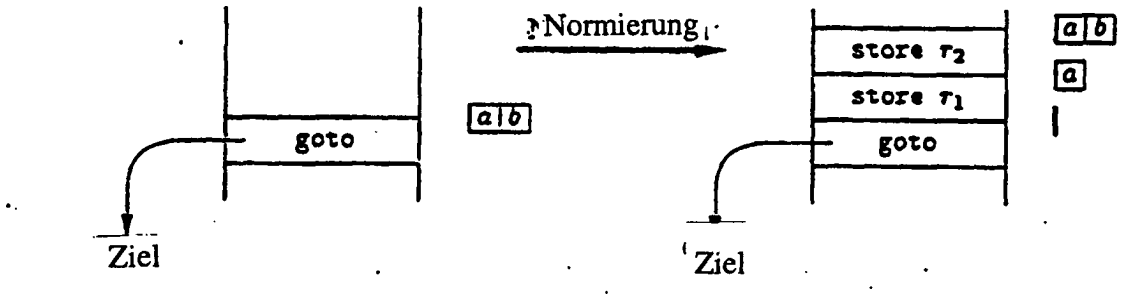
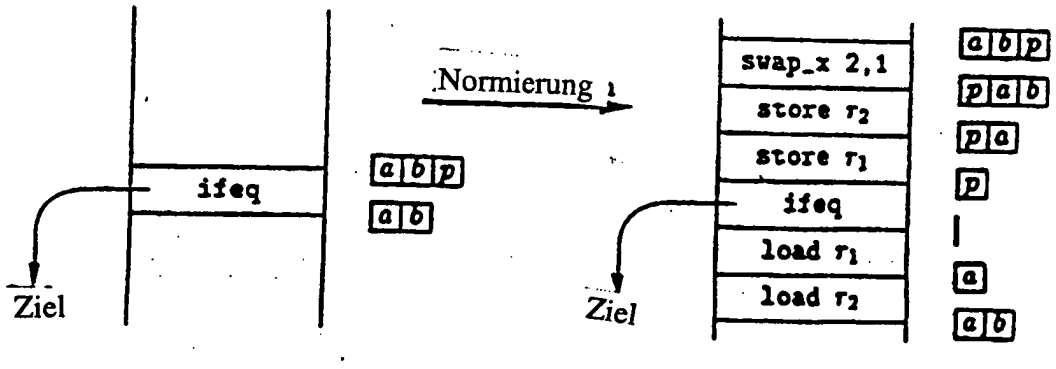


TABELLE T12
(d) Bedingte Verzweigung mit einem Argument



Patentansprüche

1. Protokoll zur Verwaltung eines ferngeladenen Programmfragmentes auf einem mitführbaren umprogrammierbaren System wie etwa einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte, wobei das Programmfragment durch einen Objektcode gebildet ist, eine Folge von Anweisungen, die von dem Mikroprozessor des mitführbaren Systems über eine virtuelle Maschine ausführbar ist, die mit einem Ausführungs-Stack und von diesen Anweisungen manipulierten lokalen Registern oder Variablen ausgestattet ist und es erlaubt, diesen Objektcode zu interpretieren, wobei das mitführbare System mit einem Endgerät verbunden ist, **dadurch gekennzeichnet**, dass das Protokoll wenigstens darin beruht, an dem mitgeführten System:

a) einen Befehl zum Fernladen dieses Programmfragmentes zu erfassen; und bei positiver Antwort auf diesen in der Erfassung eines Fernladebefehles beruhenden Schrittes

b) den dieses Programmfragment bildenden Objektcode zu lesen und diesen Objektcode zeitweilig zu speichern;

c) den gesamten zeitweilig gespeicherten Objektcode Anweisung für Anweisung einem Überprüfungsprozess zu unterziehen, wobei dieser Überprüfungsprozess in wenigstens einem Schritt der Initialisierung des Stacks der Typen auf einen leeren Zustand und der Tabelle von Typen von Registern auf einen Typ, der die Schnitt-

menge aller Typen von Daten darstellt, was einen Zustand der virtuellen Maschine zu Beginn der Ausführung des zeitweilig gespeicherten Objektcodes darstellt, und in einer Folge von Überprüfungsschritten, Anweisung für Anweisung, durch Diskriminieren der Existenz, für jede laufende Anweisung, eines Zieles, eines Zieles einer Verzweigungsanweisung, eines Zieles eines Ausnahmeverwaltungsaufrufes oder eines Zieles eines Subroutinenaufrufes, und durch Überprüfung und Aktualisierung der Wirkung der laufenden Anweisung auf den Stack der Typen und die Tabelle der Registertypen beruht, und im Fall einer gelungenen Überprüfung des Objektcodes,
d) das ferngeladene Programmfragment in einem Verzeichnis von verfügbaren Programmfragmenten aufzuzeichnen, und im Fall einer nicht gelungenen Überprüfung des Objektcodes
e) die Ausführung des Programmfragmentes auf dem mitführbaren System zu sperren.

2. Protokoll nach Anspruch 1, dadurch gekennzeichnet, dass der Schritt e) des Sperrens der Ausführung darin beruht,
f) das momentan aufgezeichnete Programmfragment zu löschen, ohne Letzteres in dem Verzeichnis von verfügbaren Programmfragmenten aufzuzeichnen, und
g) einen Fehlercode an die Leseeinrichtung zu senden.

3. Protokoll nach Anspruch 1 oder 2, dadurch gekennzeichnet, dass bei negativer Antwort auf den Schritt a), der in der Erfassung eines Fernladebefehles beruht, es darin beruht,
b') einen Befehl zur Auswahl eines in einem Verzeichnis von Programmfragmenten verfügbaren Programmfragments zu erfassen, und bei positiver Antwort auf diesen in der Erfassung eines Befehles zur Auswahl eines verfügbaren Programmfragmentes beruhenden Schritt
c') das ausgewählte verfügbare Programmfragment aufzurufen;
d') das aufgerufene verfügbare Programmfragment über die virtuelle Maschine in Abwesenheit jeglicher dynamischer Überprüfung von Variablentypen, von Zugriffsrechten auf die von dem aufgerufenen verfügbaren Programmfragment manipulierten Objekte, des Überlaufens des Ausführungsstacks bei der Ausführung jeder Anweisung auszuführen, und bei negativer Antwort auf diesen in der Erfassung eines Befehles zur Auswahl eines verfügbaren Programmfragmentes beruhenden Schritt
e') zur Verarbeitung der Standardbefehle des mitführbaren Systems überzugehen.

4. Verfahren zur Überprüfung eines ferngeladenen Programmfragmentes auf einem umprogrammierbaren mitführbaren System wie etwa einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte, wobei das Programmfragment durch einen Objektcode gebildet ist und wenigstens ein Unterprogramm, eine Folge von Anweisungen, umfasst, die von dem Mikroprozessor des mitführbaren Systems über eine virtuelle Maschine ausführbar sind, die mit einem Ausführungsstack und mit von diesen Anweisungen manipulierten Operandenregistern ausgestattet ist und es er laubt, diesen Objektcode zu interpretieren, wobei das mitgeführte System mit einer Leseeinrichtung verbunden ist, dadurch gekennzeichnet, dass das Verfahren darin beruht, nach der Erfassung eines Fernladebefehles und der Speicherung des dieses Programmfragment bildenden Objektcodes in dem überschreibbaren Speicher für jedes Unterprogramm:
α) einen Schritt der Initialisierung des Typenstacks auf einen leeren Zustand und der Tabelle der Registertypen auf einen Typ, der die Schnittmenge aller Datentypen darstellt, durchzuführen, was den Zustand der virtuellen Maschine zu Beginn der Ausführung des zeitweilig gespeicherten Objektcodes darstellt;
β) eine Überprüfung des zeitweilig gespeicherten Objektcodes, Anweisung für Anweisung, durch Diskriminierung des Vorhandenseins, für jede laufende Anweisung, eines Zieles, eines Zieles einer Verzweigungsanweisung, eines Zieles eines Ausnahmeverwaltungsaufrufes oder eines Zieles eines Subroutinenaufrufes, durchzuführen,
γ) eine Überprüfung und eine Aktualisierung der Wirkung der laufenden Anweisung auf die Datentypen des Typenstapels und die Tabelle von Registertypen in Abhängigkeit vom Vorhandensein eines Verzweigungsanweisungszieles, eines Subroutinenaufrufzieles oder eines Ausnahmeverwaltungsaufrufzieles durchzuführen, wobei die Überprüfung gelungen ist, wenn die Tabelle der Registertypen im Laufe einer Überprüfung aller Anweisungen nicht verändert wird, und der Überprüfungsprozess Anweisung für Anweisung fortgesetzt wird, bis die Tabelle der Registertypen stabil und frei von Modifikation ist, und der Überprüfungsprozess anderenfalls abgebrochen wird.

5. Überprüfungsverfahren nach Anspruch 4, dadurch gekenn zeichnet, dass die Typen von im Laufe des Überprüfungsprozesses manipulierten Variablen wenigstens umfassen:
– Klassenidentifikatoren, die den in dem Programmfragment definierten Klassen von Objekten entsprechen;
– numerische Variablentypen, die wenigstens einen Typ short, eine auf p bits codierte ganze Zahl, und einen Typ retaddr einer Rückkehradresse einer Sprunganweisung JSR umfassen;
– einen Typ null, der sich auf ungültige Objektverweise bezieht;

– einen auf Objekte bezogenen Typ object;
– einen ersten spezifischen Typ $\perp$, der die Schnittmenge aller Typen darstellt und dem Wert Null, nil, entspricht;
– einen zweiten spezifischen Typ T, der die Vereinigungsmenge aller Typen darstellt und jedem Typ von Wert entspricht.

6. Verfahren nach Anspruch 5, dadurch gekennzeichnet, dass die Menge der Variablentypen eine Untertypisierungsbeziehung erfüllt:
object $\in$ T;
short, retaddr $\in$ T;
$\perp \in$ null, short, retaddr.

7. Verfahren nach einem der Ansprüche 4 bis 6, dadurch gekennzeichnet, dass wenn die laufende Anweisung Ziel einer Verzweigungsanweisung ist, das Überprüfungsverfahren darin beruht, zu überprüfen, dass der Typenstack leer ist, und dass im Fall einer positiven Überprüfung der Überprüfungsprozess für die nächste Anweisung fortgesetzt wird und anderenfalls der Überprüfungsprozess scheitert und das Programmfragment zurückgewiesen wird.

8. Verfahren nach einem der Ansprüche 4 bis 7, dadurch gekennzeichnet, dass wenn die laufende Anweisung Ziel eines Subroutinenaufrufes ist, der Überprüfungsprozess überprüft, dass die vorhergehende Anweisung eine unbedingte Verzweigung, eine Subroutinenrückkehr oder eine Ausnahmeaufhebung darstellt, und dass im Fall einer positiven Überprüfung der Überprüfungsprozess eine Reaktualisierung des Stacks von Variablentypen durch eine Einheit vom Typ retaddr, eine Subroutinen-Rückkehradresse, vornimmt, und dass anderenfalls der Überprüfungsprozess scheitert und das Programmfragment zurückgewiesen wird.

9. Verfahren nach einem der Ansprüche 4 bis 8, dadurch gekennzeichnet, dass wenn die laufende Anweisung Ziel einer Ausnahmeverwaltung ist, der Überprüfungsprozess überprüft, dass die vorhergehende Anweisung eine unbedingte Verzweigung, eine Subroutinen-Rückkehr oder eine Ausnahmeaufhebung darstellt, und dass bei positiver Überprüfung der Überprüfungsprozess eine Reaktualisierung des Stacks der Typen durch einen Eintrag vom Typ der Ausnahmen vornimmt und dass anderenfalls der Überprüfungsprozess scheitert und das Programmfragment zurückgewiesen wird.

10. Verfahren nach einem der Ansprüche 4 bis 9, dadurch gekennzeichnet, dass wenn die laufende Anweisung Ziel einer Mehrzahl von inkompatiblen Verzweigungen ist, der Überprüfungsprozess scheitert und das Programmfragment zurückgewiesen wird.

11. Verfahren nach einem der Ansprüche 4 bis 10, dadurch gekennzeichnet, dass wenn die laufende Anweisung Ziel keiner Verzweigung ist, der Überprüfungsprozess durch Übergehen zu einer Reaktualisierung des Typenstapels fortgesetzt wird.

12. Verfahren nach einem der Ansprüche 4 bis 11, dadurch gekennzeichnet, dass der Schritt des Überprüfens der Wirkung der laufenden Anweisung auf den Typenstack wenigstens umfasst:
– einen Schritt des Überprüfens, dass der Typenausführungsstack wenigstens genausoviel Eintragungen aufweist, wie die laufende Anweisung Operanden hat;
– einen Schritt der Entnahme vom Stack und der Überprüfung, dass die Typen der Eintragungen an der Spitze des Stacks Untertypen der Typen von Operanden dieser Anweisung sind;
– einen Schritt der Überprüfung des Vorhandenseins eines ausreichenden Speicherraumes auf dem Typenstack, um die Ergebnisse der laufenden Anweisung auf den Stack legen zu können;
– einen Schritt des Legens der diesen Ergebnissen zugeordneten Datentypen auf den Stack.

13. Verfahren nach Anspruch 12, dadurch gekennzeichnet, dass wenn die laufende Anweisung eine Anweisung zum Lesen eines Registers mit Adresse n ist, der Überprüfungsprozess darin beruht,
– den Datentyp des Ergebnisses dieses Lesens durch Abfragen des Eintrages n der Tabelle der Registertypen zu überprüfen;
– die Wirkung der laufenden Anweisung auf den Typenstack durch Entnahme der den Operanden dieser laufenden Anweisung entsprechenden Einträge vom Stack und Legen des Datentyps dieses Ergebnisses auf den Stack zu bestimmen.

14. Verfahren nach Anspruch 12, dadurch gekennzeichnet, dass, wenn die laufende Anweisung eine Anweisung zum Schreiben eines Registers mit Adresse m ist, der Überprüfungsprozess darin beruht,
– die Wirkung der laufenden Anweisung auf den Typenstack und den Typ t des in dieses Register mit Adresse

m geschriebenen Operanden zu bestimmen;
– die Eintragung des Typs von der Tabelle der Registertypen an der Adresse m durch denjenigen Typ zu ersetzen, der unmittelbar höher als der zuvor gespeicherte Typ und der Typ t des in dieses Register mit Adresse m geschriebenen Operanden ist.

15. Verfahren zur Umwandlung eines Objektcodes eines Programmfragmentes, bei dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen zeigt, für jede Verzweigungsanweisung der Stack-Variablentyp in Höhe dieser Verzweigung der gleiche ist wie in Höhe der Ziele dieser Verzweigung, in einen normierten Objektcode für dieses gleiche Programmfragment, bei dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen zeigt, der Ausführungsstack bei jeder Verzweigungsanweisung und jeder Verzweigungszielanweisung leer ist, dadurch gekennzeichnet, dass das Verfahren darin beruht, für die Menge der Anweisungen des Objektcodes
– jede laufende Anweisung mit dem Datentyp des Stacks vor und nach Ausführung dieser Anweisung zu kommentieren, wobei die Kommentierungsdaten mit Hilfe einer Analyse des auf diese Anweisung bezogenen Datenstroms berechnet werden;
– in den Anweisungen und jeder laufenden Anweisung das Vorhandensein von Verzweigungen bzw. von Verzweigungszielen zu erfassen, für die der Ausführungsstack nicht leer ist, wobei die Erfassungsoperation ausgehend von den jeder laufenden Anweisung zugeteilten Stackvariablentyp-Kommentierungsdaten durchgeführt wird, und bei Erfassung eines nicht leeren Ausführungsstacks
– Stack-Variablentransfer-Anweisungen jeweils beiderseits dieser Verzweigungen bzw. dieser Verzweigungsziele einzufügen, um den Inhalt des Ausführungsstacks vor dem Verzweigen in zeitweilige Register zu entleeren und den Ausführungsstack anhand dieser zeitweiligen Register nach der Verzweigung wiederherzustellen, und anderenfalls keine Transferanweisung einzufügen, wodurch ein normierter Objektcode für dieses gleiche Programmfragment erhalten wird, bei dem der Ausführungsstack bei jeder Verzweigungsanweisung und jeder Verzweigungszielanweisung bei Nichtvorhandensein einer Modifikation der Ausführung des Programmfragmentes leer ist.

16. Verfahren zur Umwandlung eines Objektcodes eines Programmfragmentes, in dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören und ein von einer Anweisung dieses Objektcodes in ein Register geschriebener Operand von festgelegtem Typ aus diesem gleichen Register von einer anderen Anweisung dieses Objektcodes mit dem gleichen festgelegten Datentyp gelesen wird, in einen normierten Objektcode für dieses gleiche Programmfragment, bei dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, wobei ein einziger, gleicher Datentyp einem gleichen Register in dem gesamten normierten Objektcode zugeordnet ist, dadurch gekennzeichnet, dass das Verfahren darin beruht, für die Menge der Anweisungen dieses Objektcodes:
– jede laufende Anweisung mit dem Datentyp der Register vor und nach Ausführung dieser Anweisung zu kommentieren, wobei die Kommentierungsdaten mit Hilfe einer Analyse des diese Anweisung betreffenden Datenstroms berechnet werden;
– eine Neuzuteilung der Register durch Erfassen der mit verschiedenen Typen verwendeten Ursprungsregister, Aufteilen dieser Ursprungsregister auf unterschiedliche normierte Register, ein normiertes Register für jeden verwendeten Datentyp, und eine Reaktualisierung der Anweisungen durchzuführen, die die Operanden manipulieren, die auf diese normierten Register zugreifen.

17. Verfahren nach Anspruch 15, dadurch gekennzeichnet, dass der Schritt, der darin beruht, in den Anweisungen und jeder laufenden Anweisung das Vorhandensein von Verzweigungen bzw. von Verzweigungszielen zu erfassen, für die der Ausführungsstack nicht leer ist, nach der Erfassung jeder Anweisung von entsprechendem Rang i darin beruht,
– jeder Anweisung von Rang i eine Menge von neuen Registern zuzuordnen, wobei in Höhe dieser Anweisung jeder aktiven Stackvariable ein neues Register zugeordnet wird;
– jede erfasste Anweisung vom Rang i zu untersuchen und das Vorhandensein eines Verzweigungszieles bzw. einer Verzweigung zu diskriminieren und in dem Fall, wo die Anweisung vom Rang i ein Verzweigungsziel ist und der Ausführungsstack in Höhe dieser Anweisung nicht leer ist,
– für jede vorhergehende Anweisung vom Rang i – 1, die durch eine Verzweigung, eine Ausnahmeaufhebung oder eine Programmrückkehr gebildet ist, wenn die erfasste Anweisung vom Rang i nur durch eine Verzweigung zugänglich ist,
– eine Menge von load-Anweisungen ausgehend von der Menge von neuen Registern vor die erfasste Anweisung vom Rang i einzufügen, wobei alle Verzweigungen zu der erfassten Anweisung vom Rang i auf die erste eingefügte Ladeeinweisung load umgeleitet werden, und
– für jede die Folge fortsetzende vorhergehende Anweisung vom Rang i - 1, wenn die erfasste Anweisung vom

Rang i sowohl über eine Verzweigung als auch über die vorhergehende Anweisung vom Rang i - 1 erreichbar ist,
– eine Menge von Speicheroperationen store in die Menge von neuen Registern vor die erfasste Anweisung vom Rang i und eine Menge von Ladebefehlen load ausgehend von dieser Menge von neuen Registern einzufügen, wobei alle Verzweigungen zu der erfassten Anweisung vom Rang i auf die erste eingefügte Ladeanweisung load umgeleitet werden, und in dem Fall, wo die erfasste Anweisung vom Rang i eine Verzweigung zu einer festgelegten Anweisung ist, und
– für jede erfasste Anweisung vom Rang i, die durch eine unbedingte Verzweigung gebildet ist,
– vor diese erfasste Anweisung vom Rang i eine Mehrzahl von Speicheranweisungen store einzufügen, wobei jedem neuen Register eine Speicheranweisung zugeordnet ist; und
– für jede erfasste Anweisung vom Rang i, die durch eine bedingte Verzweigung gebildet ist, und für eine Zahl m > 0 von durch diese bedingte Verzweigungsanweisung manipulierten Operanden,
– vor diese erfasste Anweisung vom Rang i eine Permutationsanweisung swap-x an der Spitze des Ausführungsstacks der m Operanden der erfassten Anweisung vom Rang i und der n folgenden Werte einzufügen, wobei diese Permutationsoperation es erlaubt, die n in der Menge von neuen Registern zu speichernden Werte an die Spitze des Ausführungsstacks zurückzubringen, und
– vor die Anweisung vom Rang i eine Menge von Speicheroperationen store zum Speichern in der Menge der neuen Register einzufügen und
– nach der erfassten Anweisung vom Rang i eine Menge von Ladeanweisungen load zum Laden aus der Menge von neuen Registern einzufügen.

18. Verfahren nach Anspruch 16, dadurch gekennzeichnet, dass der Schritt, der darin beruht, eine Neuzuweisung der Register durch Erfassen der mit den verschiedenen Typen verwendeten ursprünglichen Register vorzunehmen, darin beruht,
– die Lebensdauerintervalle jedes Registers zu bestimmen;
– den Hauptdatentyp jedes Lebensdauerintervalls zu bestimmen, wobei der Hauptdatentyp eines Lebensdauerintervalls j für ein Register r definiert ist durch die obere Grenze der in diesem Register r von den zu dem Lebensdauerintervall j gehörenden Speicheroperationen store gespeicherten Datentypen;
– einen Graph von Interferenzen zwischen den Lebensdauerintervallen aufzustellen, wobei dieser Graph von Interferenzen aus einem nicht-orientierten Graphen besteht, von dem jeder Knoten durch ein Lebensdauerintervall gebildet ist, und dessen Kanten zwischen zwei Knoten j_1 und j_2 existieren, wenn ein Knoten eine an das Register des anderen Knotens adressierte Speicheroperation enthält oder umgekehrt;
– die Eindeutigkeit eines jedem Register zugeordneten Datentyps in dem Interferenzgraphen zu übersetzen durch Einfügen von Kanten zwischen jedem Knotenpaar des Interferenzgraphen, so lange nicht zwei Knoten eines Paares von Knoten den gleichen zugeordneten Hauptdatentyp haben;
– eine Instanzierung des Interferenzgraphen durch Zuweisen einer Registernummer zu jedem Lebensdauerintervall derart durchzuführen, dass zwei in dem Interferenzgraphen benachbarten Lebensdauerintervallen unterschiedliche Registernummern zugeordnet werden.

19. Mitführbares, durch Fernladen von Programmfragmenten umprogrammierbares System mit wenigstens einem Mikroprozessor, einem Schreib-Lesespeicher, einem Ein-/Ausgabemodul, einem elektrisch umprogrammierbaren nichtflüchtigen Speicher und einem permanenten Speicher, in dem ein Hauptprogramm und eine virtuelle Maschine abgelegt sind, die die Ausführung des Hauptprogramms und wenigstens eines Programmfragmentes über den Mikroprozessor ermöglicht, dadurch gekennzeichnet, dass das mitführbare System wenigstens ein Programmmodul zur Verwaltung und Überprüfung eines leeren Zustandes eines Typenstacks für jede Verzweigungsanweisung eines ferngeladenen Programmfragmentes umfasst, wobei das Programm zur Verwaltung und Überprüfung im permanenten Speicher abgelegt ist.

20. Mitführbares, durch Fernladen von Programmfragmenten umprogrammierbares System mit wenigstens einem Mikroprozessor, einem Schreib-Lesespeicher, einem Ein-/Ausgabemodul, einem elektrisch umprogrammierbaren nichtflüchtigen Speicher und einem permanenten Speicher, in dem ein Hauptprogramm und eine virtuelle Maschine abgelegt sind, die die Ausführung des Hauptprogramms und wenigstens eines Programmfragmentes über den Mikroprozessor ermöglicht, dadurch gekennzeichnet, dass das mitführbare System wenigstens ein Programm zur Verwaltung und Überprüfung nach dem Protokoll zur Verwaltung eines ferngeladenen Programmfragmentes nach einem der Ansprüche 1 bis 3 umfasst, wobei das Verwaltungs- und Überprüfungs-Programmmodul im permanenten Speicher abgelegt sind.

21. Mitführbares System nach Anspruch 20, dadurch gekennzeichnet, dass es wenigstens ein Unterprogrammmodul zur Überprüfung eines ferngeladenen Programmfragmentes nach dem Überprüfungsprozess nach einem der Ansprüche 4 bis 14 umfasst.

22. System zur Umwandlung eines Objektcodes eines Programmfragmentes, in dem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen aufweist, für jede Verzweigungsanweisung der Stack-Variablentyp in Höhe dieser Verzweigung der gleiche wie in Höhe der Ziele dieser Verzweigung ist, und ein durch eine Anweisung dieses Objektcodes in ein Register geschriebener Operand von festgelegtem Typ aus diesem gleichen Register von einer anderen Anweisung dieses Objektcodes mit dem gleichen festgelegten Datentyp erneut gelesen wird, in einen normierten Objektcode für dieses gleiche Programmfragment, in welchem die Operanden jeder Anweisung zu den von dieser Anweisung manipulierten Datentypen gehören, der Ausführungsstack kein Überlaufen aufweist, der Ausführungsstack bei jeder Verzweigungsanweisung und jeder Verzweigungszielanweisung leer ist, ein einziger, gleicher Datentyp einem gleichen Register in dem gesamten normierten Objektcode zugewiesen ist, dadurch gekennzeichnet, dass das Umwandlungssystem wenigstens, abgelegt im Arbeitsspeicher eines Entwicklungscomputers oder einer Workstation, ein Programmmodul zur Umwandlung dieses Objektcodes in einen normierten Objektcode nach dem Verfahren gemäß einem der Ansprüche 15 bis 18 umfasst, was es erlaubt, einen normierten Objektcode für das Programmfragment zu erzeugen, der den Kriterien zur Überprüfung dieses ferngeladenen Programmfragmentes genügt.

23. Direkt in den internen Speicher eines mitführbaren, umprogrammierbaren Systems wie etwa einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte ladbares Computerprogrammprodukt, wobei dieses mitführbare System das Fernladen eines Programmfragmentes erlaubt, das durch einen Objektcode gebildet ist, eine Folge von Anweisungen, die von dem Mikroprozessor des mitführbaren Systems über eine virtuelle Maschine ausführbar ist, die mit einem Ausführungsstack und mit von diesen Anweisungen manipulierten lokalen Registern oder Variablen ausgestattet ist und die es erlaubt, diesen Objektcode zu interpretieren, wobei dieses Computerprogrammprodukt Objektcodeabschnitte für die Ausführung des Protokolls zur Verwaltung eines ferngeladenen Programmfragmentes auf dem mitführbaren System nach einem der Ansprüche 1 bis 3 aufweist, wenn dieses mitführbare System mit einem Endgerät verbunden ist, und das Programm von dem Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine ausgeführt wird.

24. Direkt in den internen Speicher eines mitführbaren umprogrammierbaren Systems wie etwa einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte ladbares Computerprogrammprodukt, wobei dieses mitführbare System das Fernladen eines Programmfragmentes erlaubt, das durch einen Objektcode gebildet ist, eine Folge von Anweisungen, die von dem Mikroprozessor des mitführbaren Systems über eine virtuelle Maschine ausführbar sind, die mit einem Ausführungsstack und mit von diesen Anweisungen manipulierten Operandenregistern ausgestattet sind und die es erlaubt, diesen Objektcode zu interpretieren, wobei dieses Computerprogrammprodukt Objektcodeabschnitte zur Ausführung der Schritte zur Überprüfung eines ferngeladenen Programmfragmentes auf dem mitführbaren System nach einem der Ansprüche 4 bis 14 umfasst, wenn das mitführbare System mit einem Endgerät verbunden ist, und das Programm von dem Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine ausgeführt wird.

25. Computerprogrammprodukt mit Objektcodeabschnitten zur Ausführung der Schritte des Verfahrens zur Umwandlung eines Objektcodes eines ferngeladenen Programmfragmentes in einen normierten Objektcode für dieses gleiche Programmfragment nach einem der Ansprüche 15 bis 18.

26. Computerprogrammprodukt, aufgezeichnet auf einem in einem umprogrammierbaren mitführbaren System verwendbaren Träger wie etwa einer mit einem überschreibbaren Speicher ausgestatteten Mikroprozessorkarte, wobei dieses mitführbare System das Fernladen eines Programmfragmentes erlaubt, das durch einen Objektcode gebildet ist, eine Folge von Anweisungen, die durch den Mikroprozessor des mitführbaren Systems über eine virtuelle Maschine ausführbar sind, die mit einem Ausführungsstack und mit von diesen Anweisungen manipulierten lokalen Registern oder Variablen ausgestattet ist, und die es erlaubt, diesen Objektcode zu interpretieren, wobei dieses Computerprogrammprodukt wenigstens umfasst:
– durch den Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine lesbare Programm-Mittel zum Steuern der Ausführung einer Prozedur zur Verwaltung des Fernladens eines ferngeladenen Programmfragmentes;
– durch den Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine lesbare Programm-Mittel zum Steuern der Ausführung einer Prozedur der Überprüfung, Anweisung für Anweisung, eines leeren Zustandes des Typenstacks für jede Verzweigungsanweisung des das Programmfragment bildenden Objektcodes;
– durch den Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine lesbare Programm-Mittel zum Steuern der Ausführung eines ferngeladenen Programmfragmentes nach einer oder in Abwesenheit einer Umwandlung des Objektcodes dieses Programmfragmentes in normierten Objektcode für dieses gleiche Programmfragment.

27. Computerprogrammprodukt nach Anspruch 26, ferner mit durch den Mikroprozessor dieses mitführbaren Systems über die virtuelle Maschine lesbaren Programm-Mitteln zum Steuern der Sperrung der Ausführung, in dem mitführbaren System, des Programmfragmentes im Fall einer misslungenen Überprüfungsprozedur dieses Programmfragmentes.

Es folgen 14 Blatt Zeichnungen

Anhängende Zeichnungen

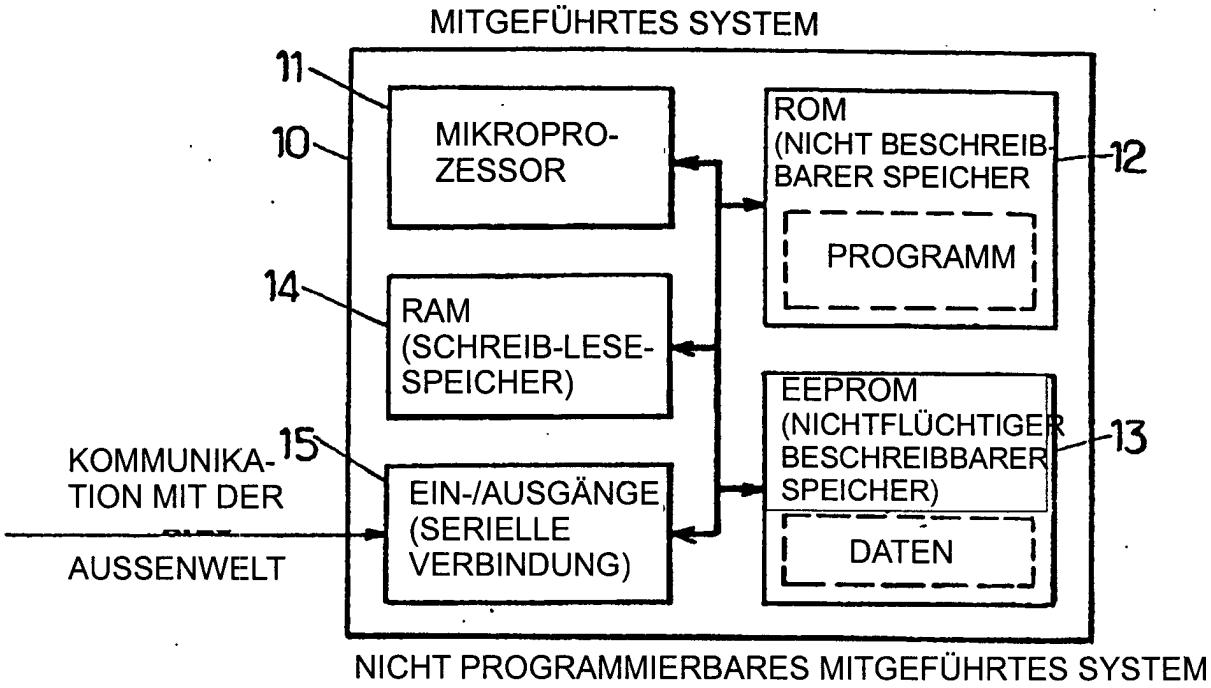


FIG.1a. (STAND DER TECHNIK)

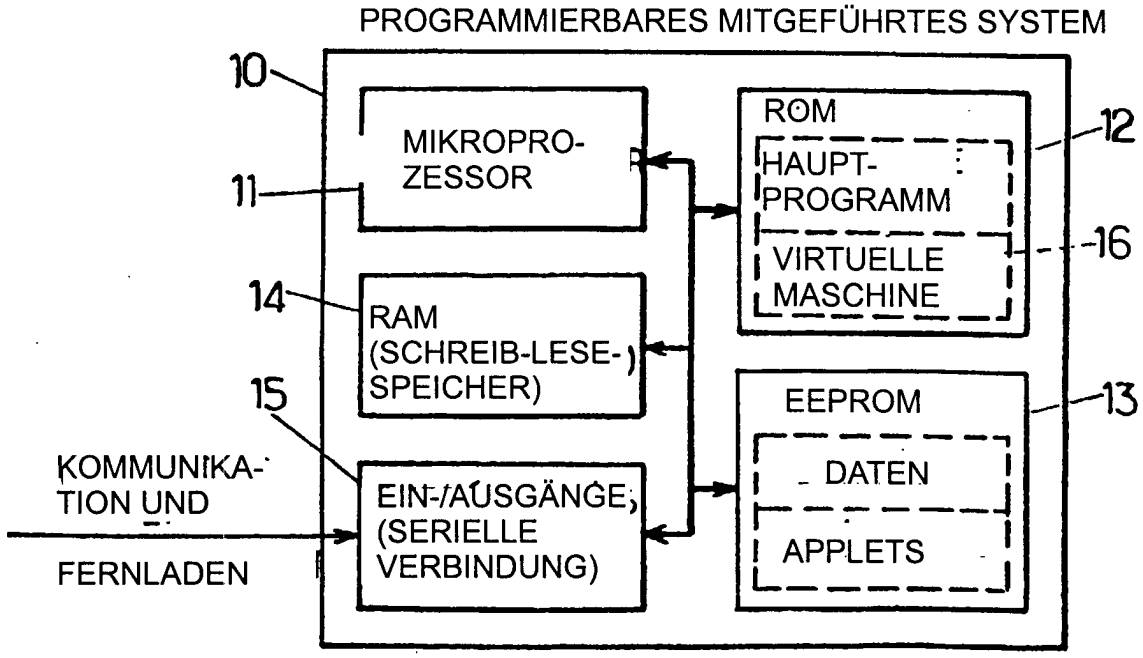


FIG.1b. (STAND DER TECHNIK)

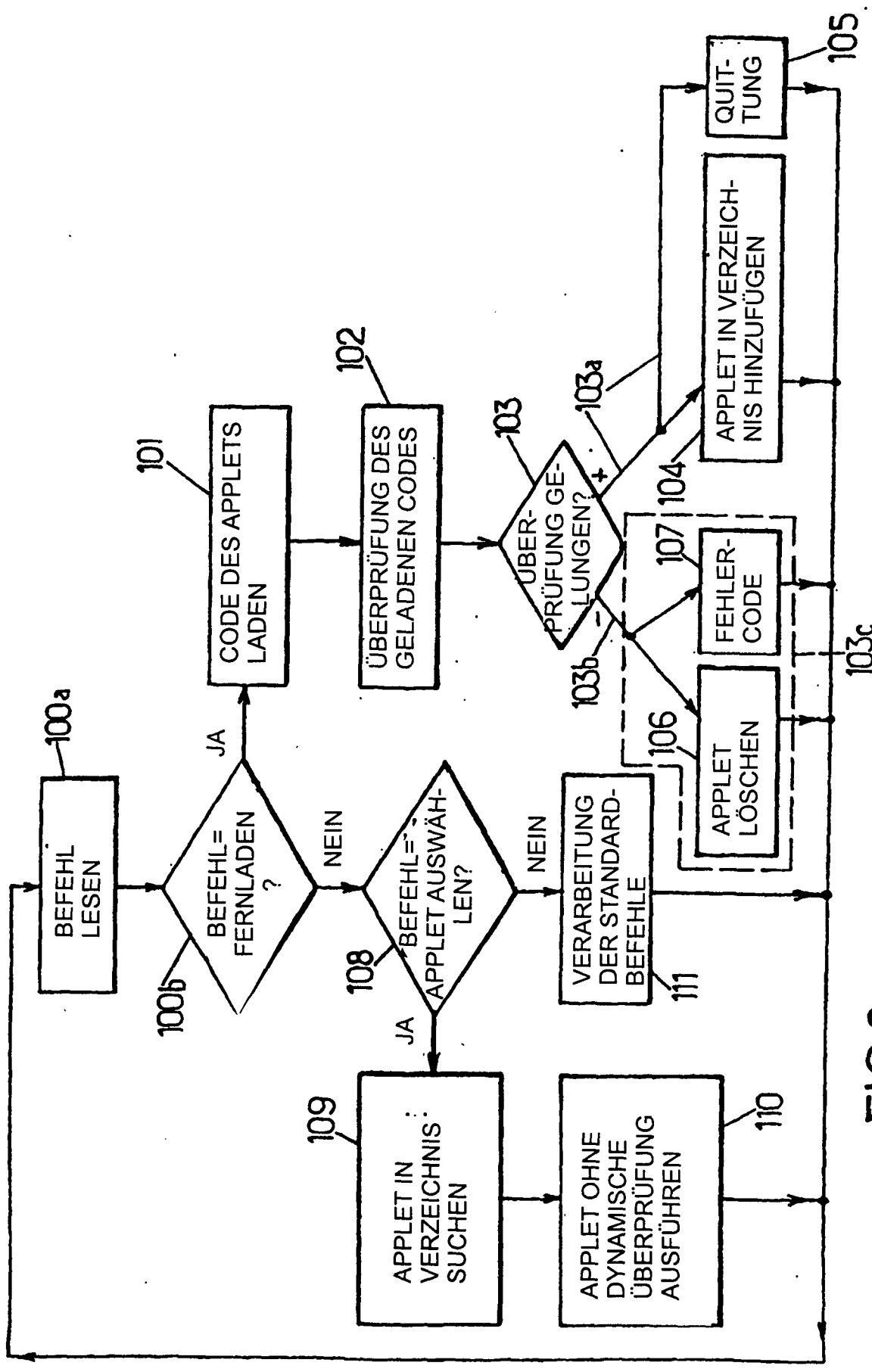


FIG.2. VERWALTUNGSPROTOKOLL

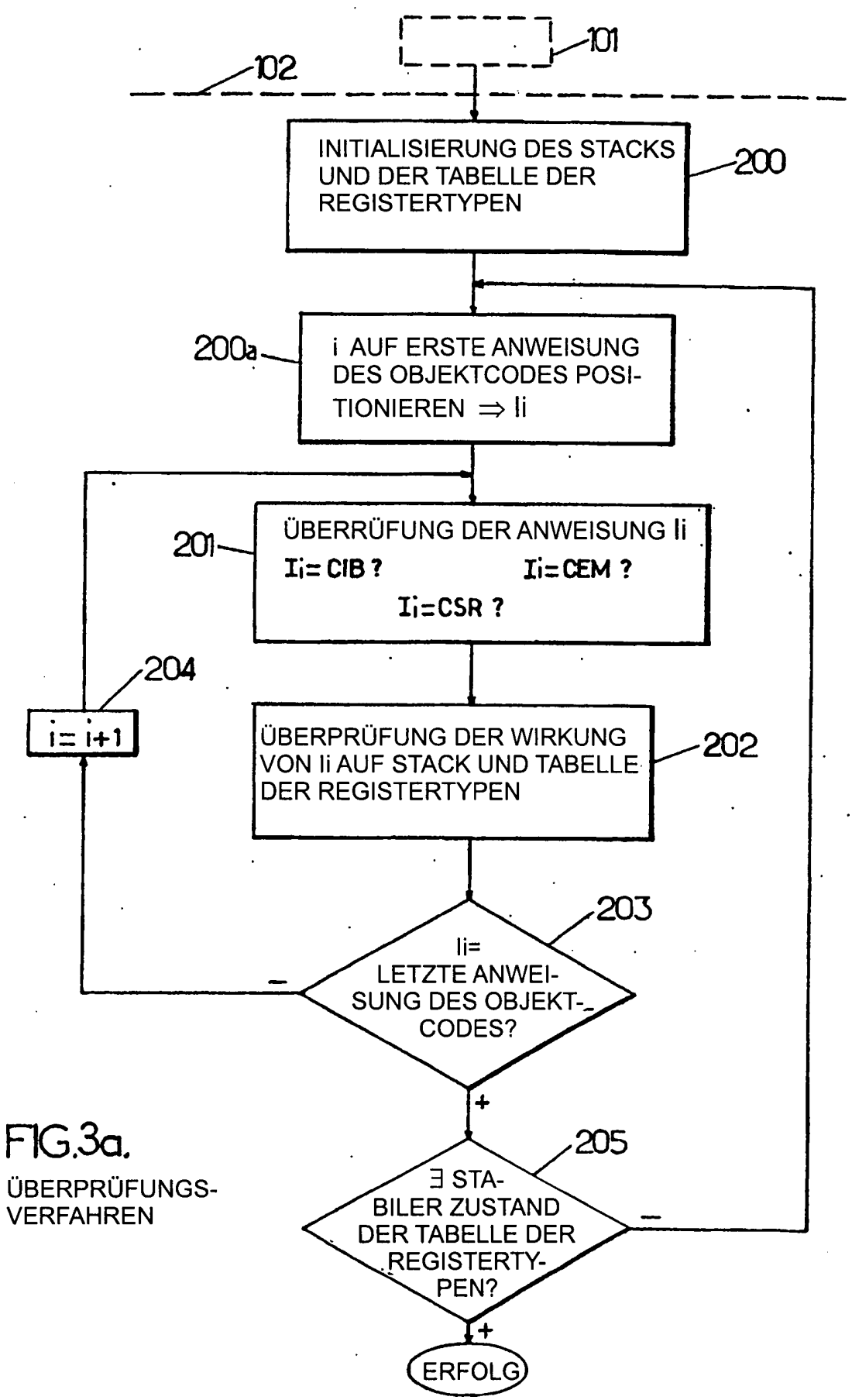


FIG.3a.

ÜBERPRÜFUNGS-VERFAHREN

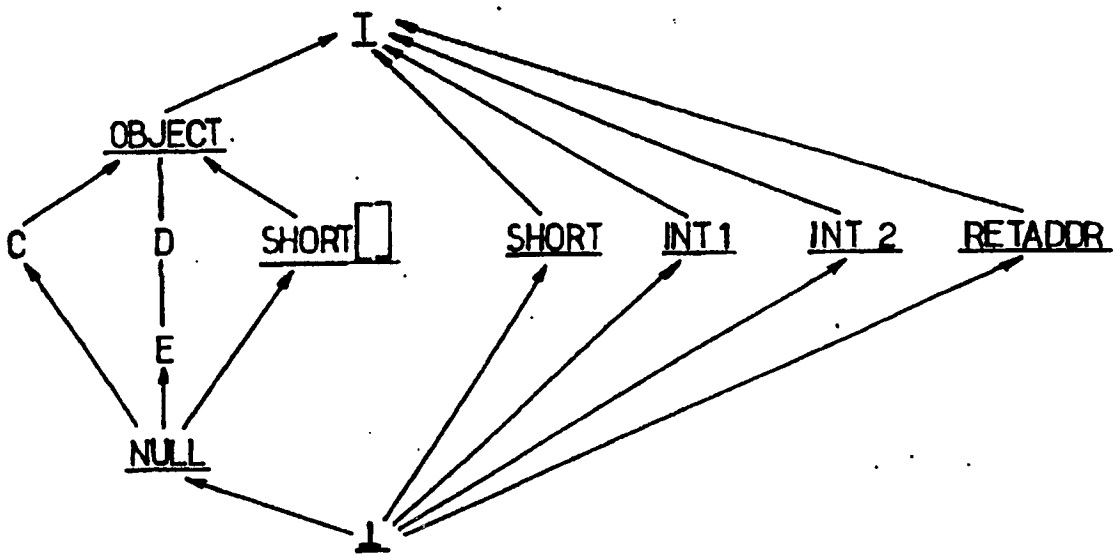


FIG.3b. DATENTYPEN UND UNTERTYPISIERUNGS-BEZIEHUNG

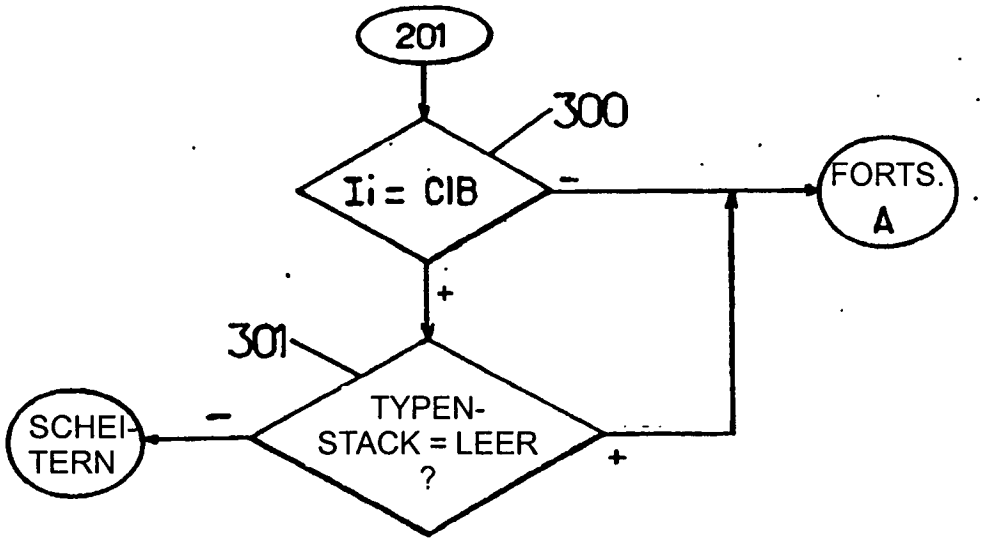


FIG.3c. ÜBERPRÜFUNG: VERWALTUNG EINER VERZWEIGUNGSANWEISUNG

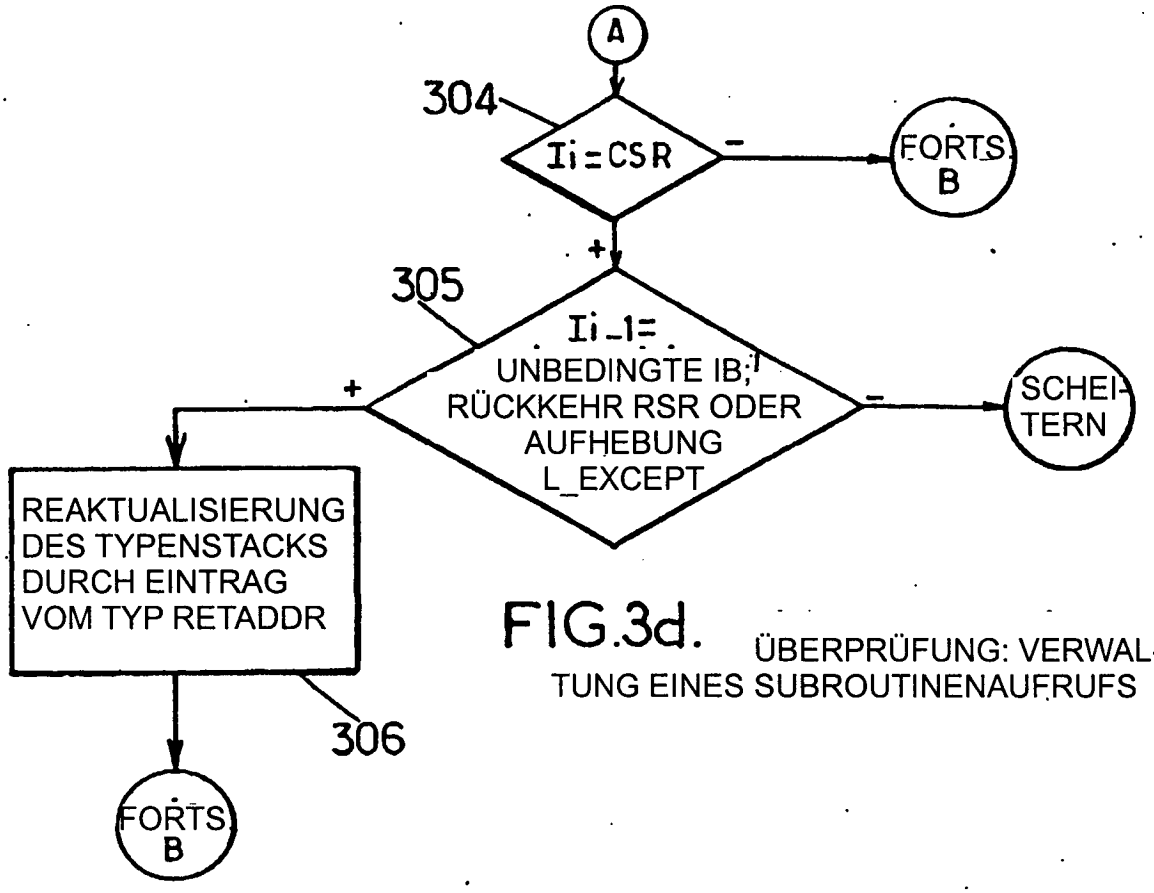


FIG.3d. ÜBERPRÜFUNG: VERWALTUNG EINES SUBROUTINENAUFRUFS

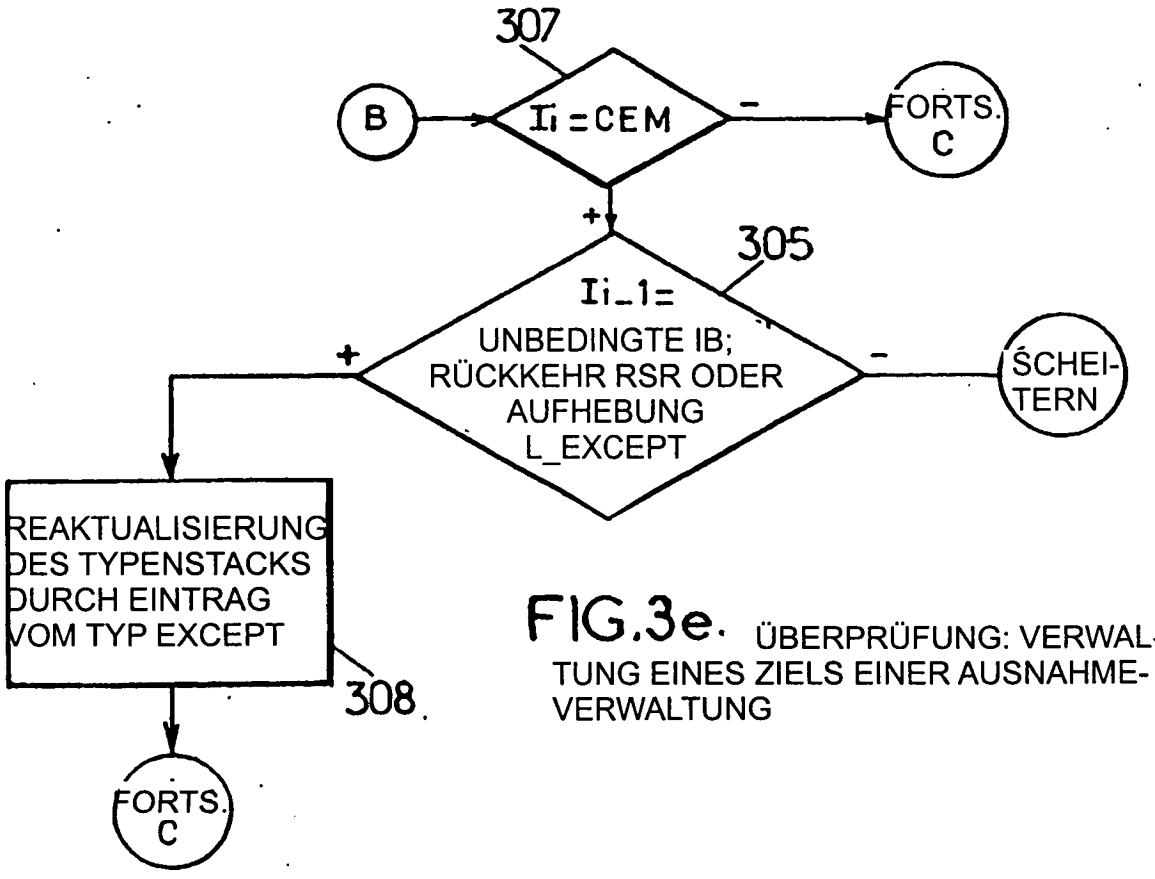


FIG.3e. ÜBERPRÜFUNG: VERWALTUNG EINES ZIELS EINER AUSNAHMEVERWALTUNG

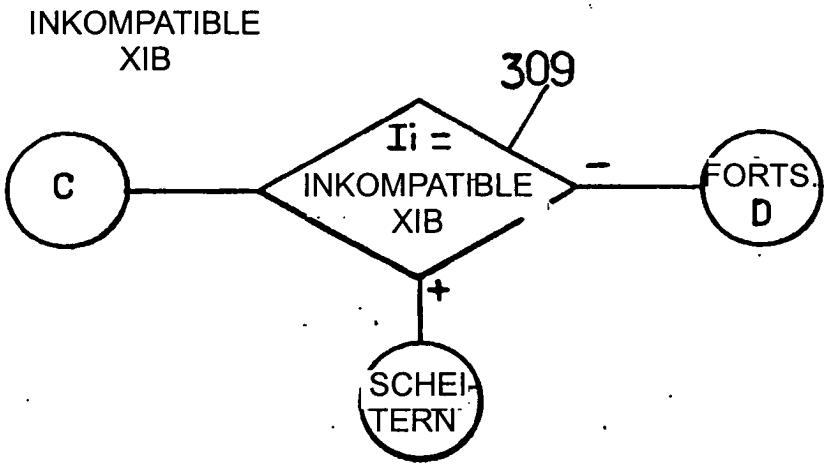


FIG.3f. ÜBERPRÜFUNG: VERWALTUNG DES ZIELS VON INKOMPATIBLEN VERZWEIGUNGEN

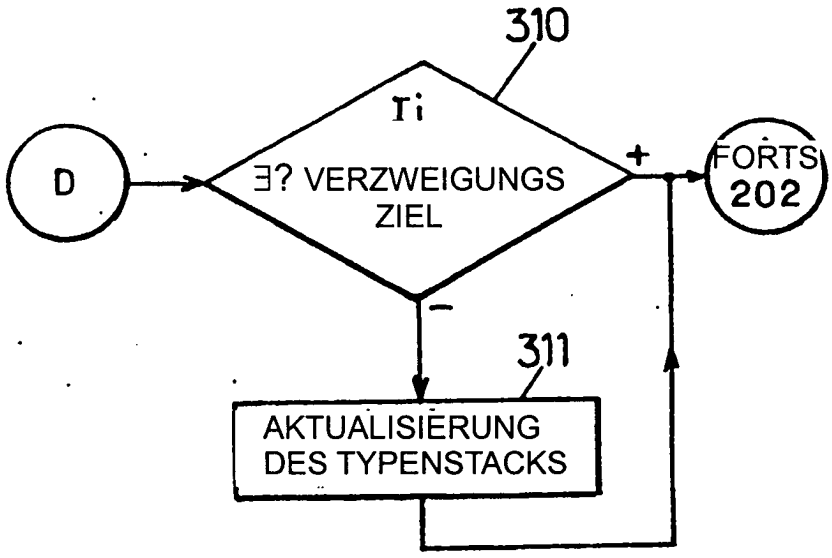


FIG.3g. ÜBERPRÜFUNG: VERWALTUNG DES FEHLENS EINES VERZWEIGUNGSZIELS

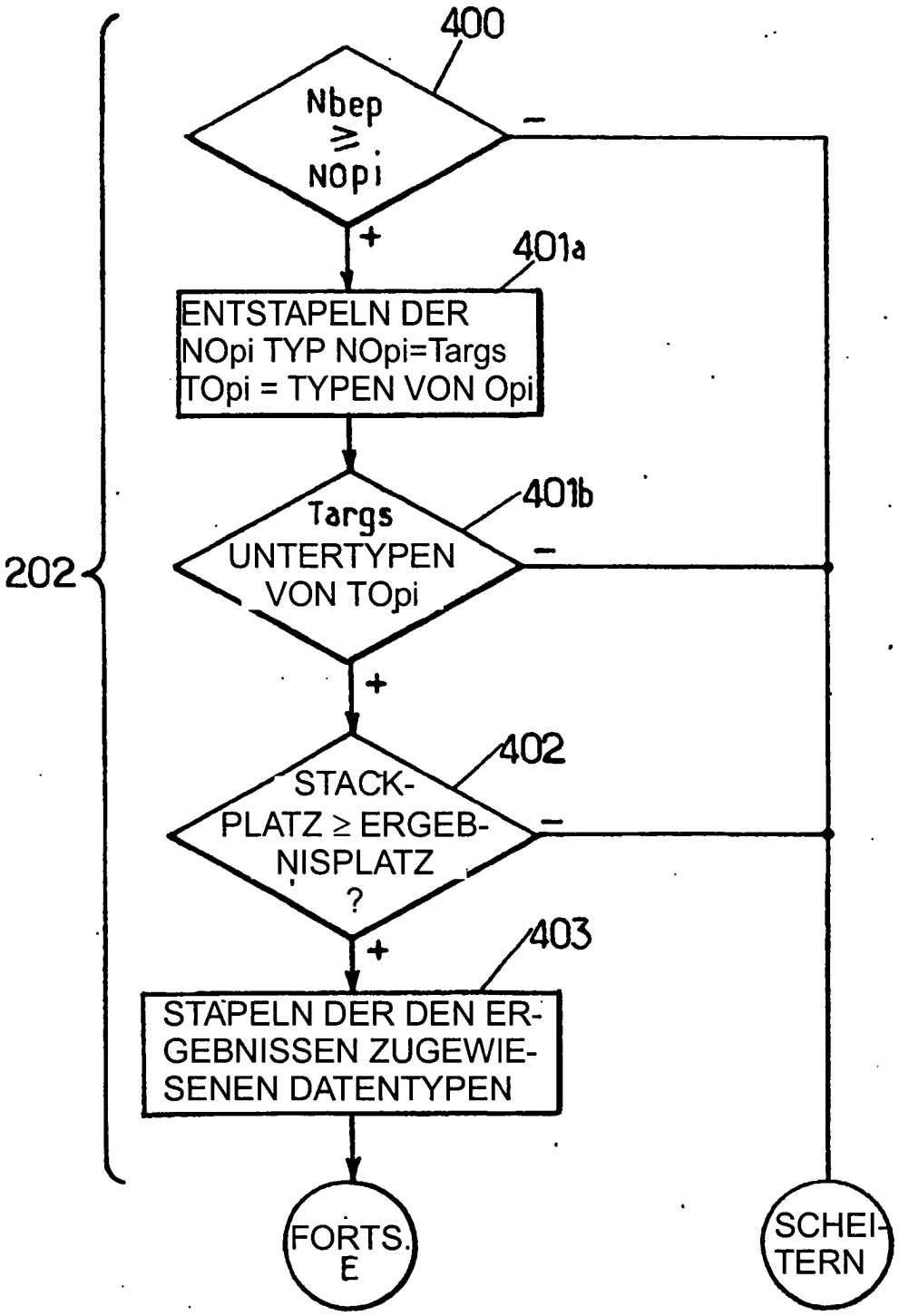


FIG.3h. ÜBERPRÜFUNG: WIRKUNG DER LAUFENDEN ANWEISUNG AUF TYPENSTACK

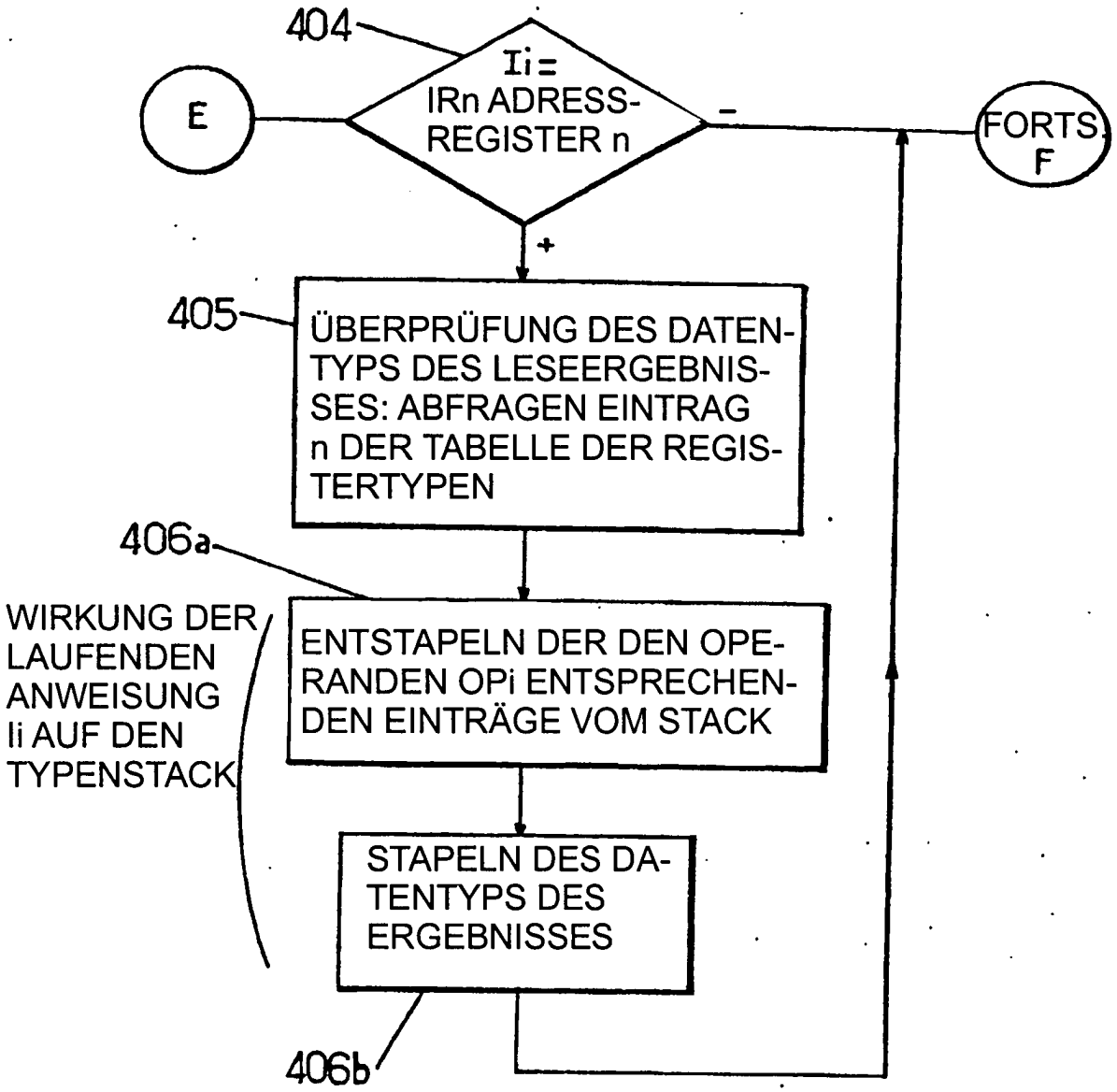


FIG.3i. ÜBERPRÜFUNG: VERWALTUNG EINER REGISTER-LESEANWEISUNG

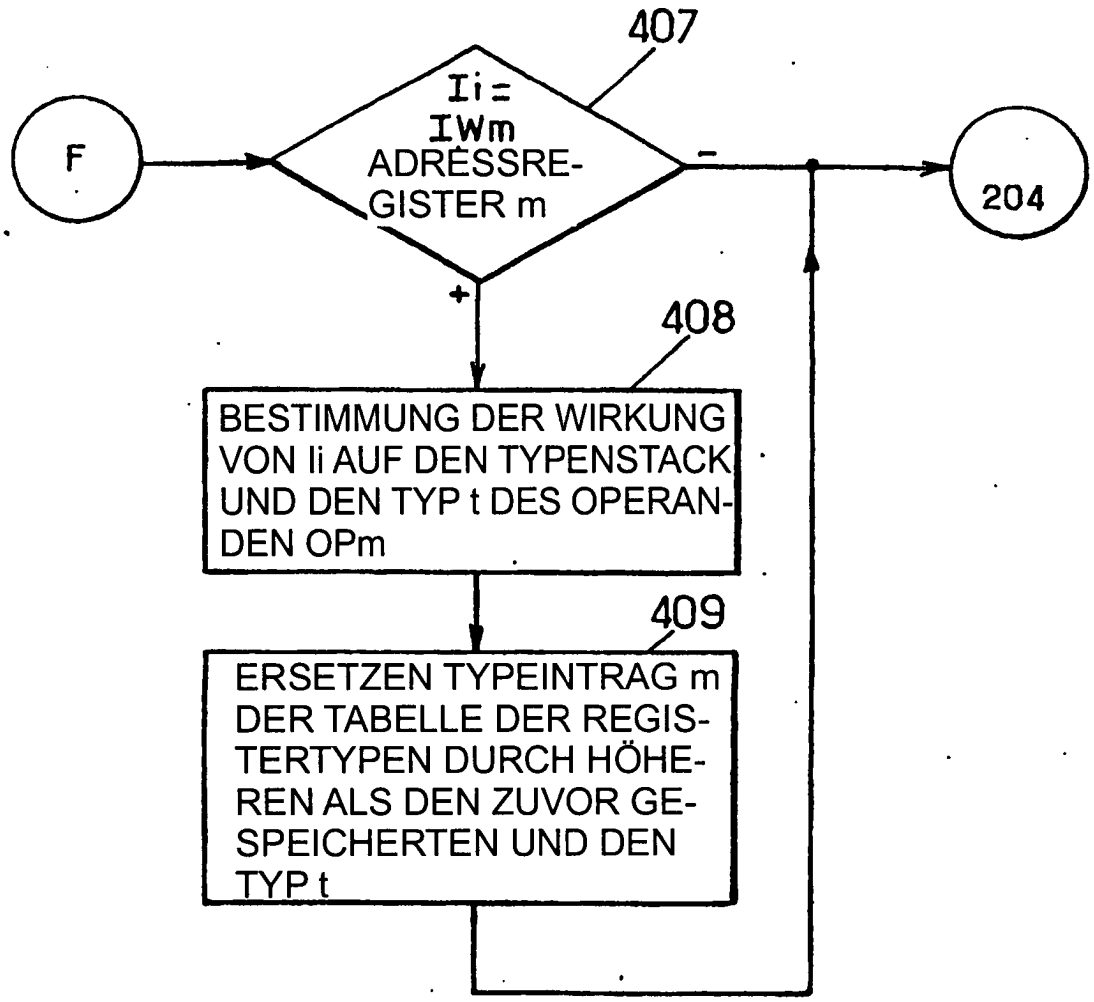


FIG.3j. ÜBERPRÜFUNG. VERWALTUNG EINER ANWEISUNG ZUM SCHREIBEN EINES REGISTERS

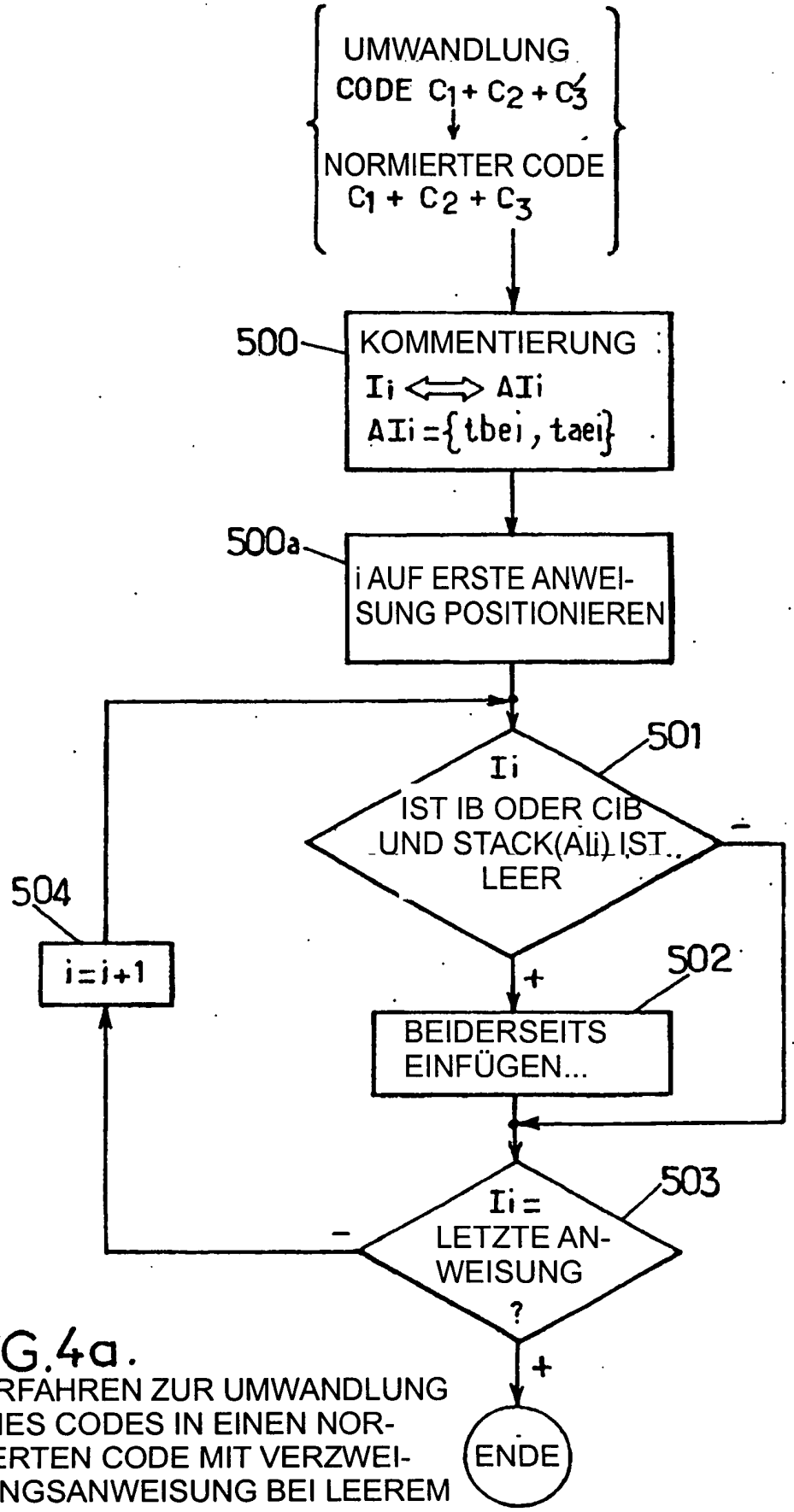


FIG.4a.

VERFAHREN ZUR UMWANDLUNG EINES CODES IN EINEN NORMIERTEN CODE MIT VERZWEIGUNGSANWEISUNG BEI LEEREM STACK

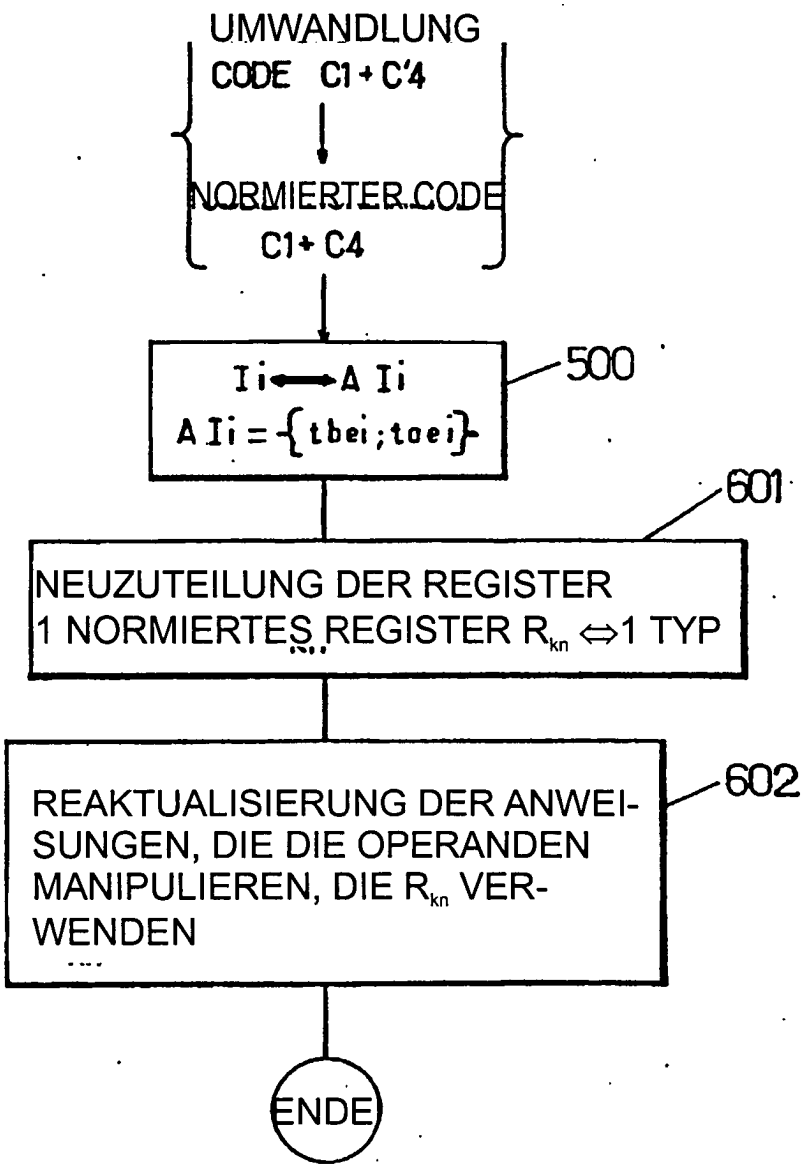


FIG.4b. VERFAHREN ZUR UMWANDLUNG EINES CODES IN EINEN NORMIERTEN CODE, DER TYPENREGISTER VERWENDET

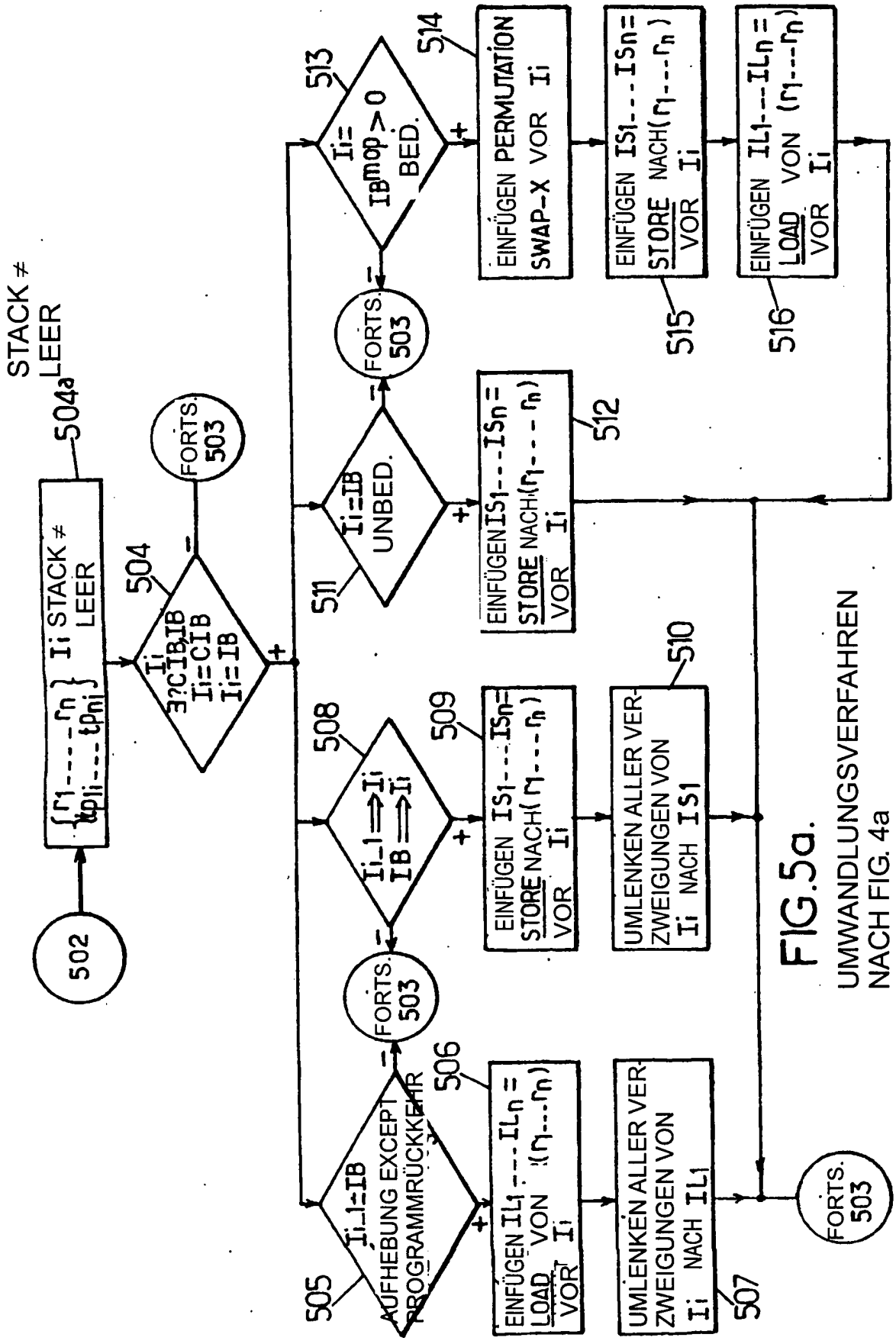


FIG.5a.
UMWANDLUNGSVERFAHREN
NACH FIG. 4a

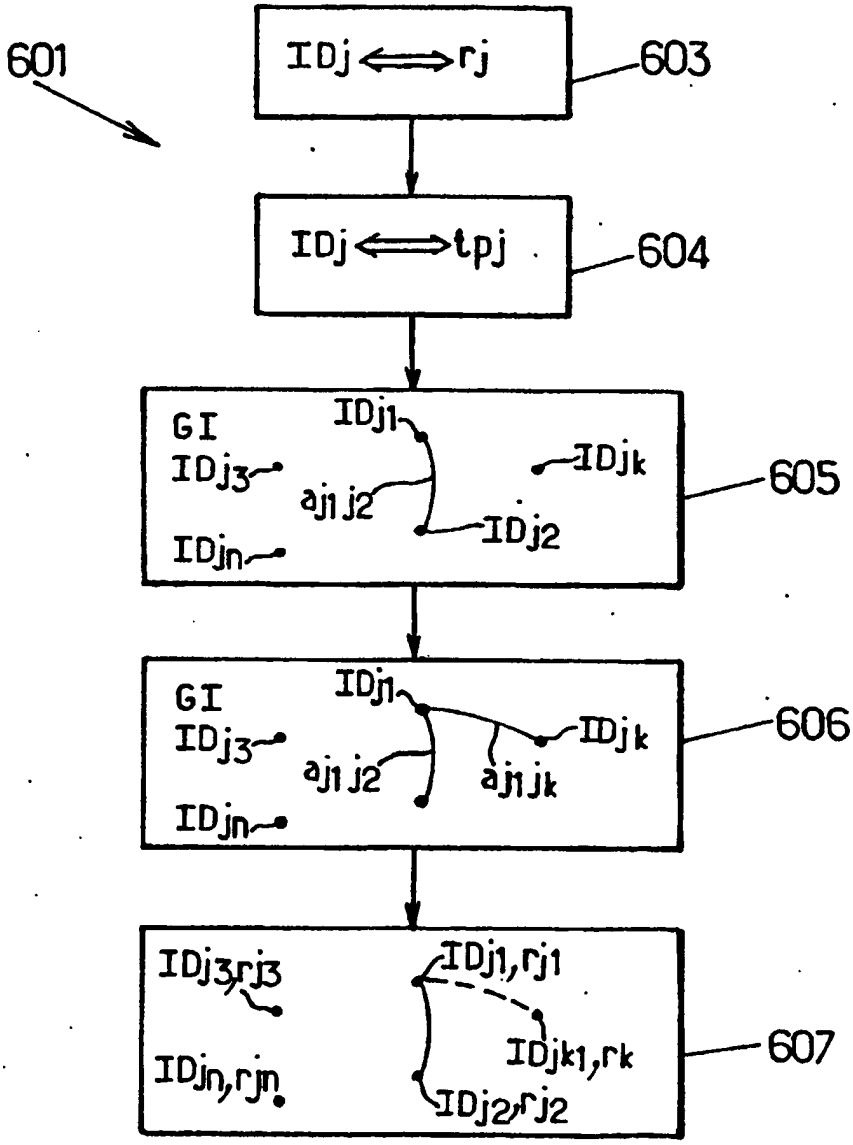


FIG.5b.

UMWANDLUNGSVERFAHREN
NACH FIG. 4b

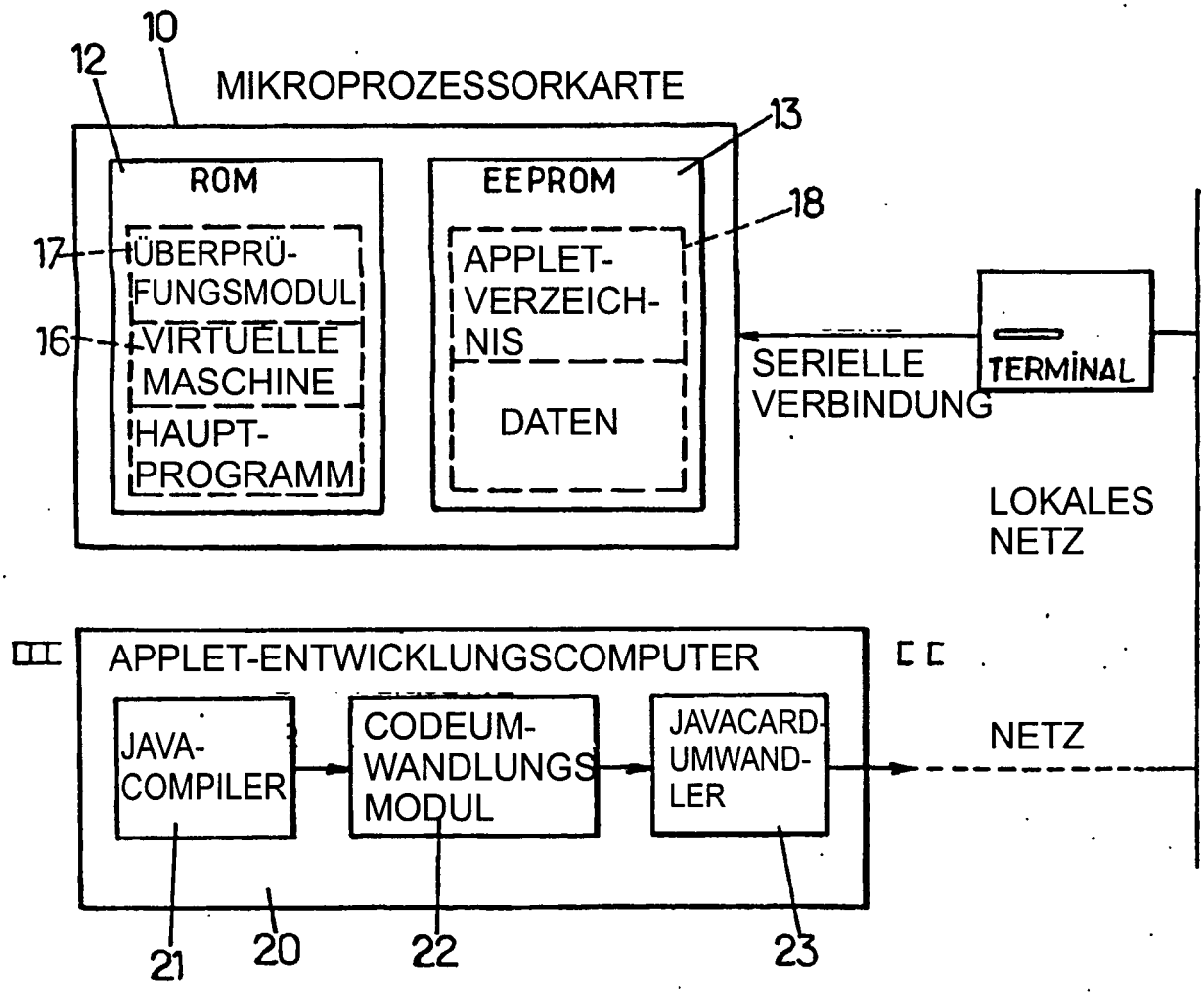

10
12
MIKROPROZESSORKARTE
13
ROM
EEPROM
18
17
ÜBERPRÜ-
FUNGSMODUL
16
VIRTUELLE
MASCHINE
HAUPT-
PROGRAMM
APPLET-
VERZEICH-
NIS
DATEN
SERIELLE
VERBINDUNG
TERMINAL
LOKALES
NETZ
APPLET-ENTWICKLUNGSCOMPUTER
JAVA-
COMPILER
CODEUM-
WANDLUNGS
MODUL
JAVACARD-
UMWAND-
LER
NETZ
21
20
22
23

FIG. 6.