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

(10) **DE 694 32 974 T2** 2004.05.27

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 699 320 B1**
(21) Deutsches Aktenzeichen: **694 32 974.6**
(86) PCT-Aktenzeichen: **PCT/US94/05182**
(96) Europäisches Aktenzeichen: **94 916 719.1**
(87) PCT-Veröffentlichungs-Nr.: **WO 94/027213**
(86) PCT-Anmeldetag: **10.05.1994**
(87) Veröffentlichungstag der PCT-Anmeldung: **24.11.1994**
(97) Erstveröffentlichung durch das EPA: **06.03.1996**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **23.07.2003**
(47) Veröffentlichungstag im Patentblatt: **27.05.2004**

(51) Int Cl.[7]: **G06F 11/36**

(30) Unionspriorität:
59208 **10.05.1993** **US**

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

(73) Patentinhaber:
Thinking Software, Inc., Cupertino, Calif., US

(72) Erfinder:
Shapiro, Benjamin V., West Caldwell, US

(74) Vertreter:
Mitscherlich & Partner, Patent- und Rechtsanwälte, 80331 München

(54) Bezeichnung: **VERFAHREN UND VORRICHTUNG ZUR AUTOMATISCHEN ANALYSE EINES ZIELPROGRAMMS**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

Hintergrund der Erfindung

Gebiet der Erfindung:

[0001] Diese Erfindung bezieht sich auf ein Verfahren zur statischen und dynamischen Analyse von Computerprogrammen, das es gestattet, eine Unsicherheit über den Ort von Fehler(n) zu minimieren, welche die Quelle der Programmversager wie auch Plätze für notwendige Modifikationen im Falle von veränderten Spezifikationen sind.
[0002] Die erfundenen Verfahren stellen auch eine originale Schnittstelle auf sehr hohem Niveau zwischen dem Anwender eines Zielcomputersoftwaresystems und seinen internen Prozessen bereit, wobei die Schnittstelle eine sehr effektive Methode des Aktivierens der Zielsoftwareanalyse gestattet.
[0003] Dieselbe Schnittstelle gestattet die Einführung einer Programmiersprache auf sehr hohem Niveau für Systemmodifikationen und Vorwärtsentwicklung, was wir resultatorientierte Programmierung (ROP) nennen.
[0004] Die folgende Beschreibung stellt auch eine Alternative zur traditionellen Form von Computersoftwareprozessobjektcode durch Vorschlagen eines eX-Objektcodes bereit, wobei eX für EXperten-Programm-Diagnose steht.
[0005] eX-Objektcode ist für die Programmanalyse, sowohl statisch als auch dynamisch, und für die Programmwartung geeigneter. Es vereinfacht und manchmal eliminiert es das Neucompilieren und Neuverknüpfen (Linken) des Zielprozesscodes als Ergebnis der Modifikationen dieses Prozesses.

Beschreibung verwandten Stands der Technik

[0006] Mehrere Dekaden der Computersoftwareentwicklung haben gezeigt, dass:

a: Es keinen automatischen Weg gibt, um zu beweisen, dass ein Computerprogrammprozess immer das korrekte oder gewünschte Resultat ergeben wird;

b: Es daher keine "fehlerfreie" Software gibt, d. h. die Software ist nur bis zu dem Grade "korrekt", in dem sie getestet wurde, unter der Annahme, dass das Testen korrekt war;

c: Ein erschöpfendes Testen ist nur bei der einfachsten Software machbar;

d: Derzeitige Softwareentwicklung stellt nichts bereit, was analog zu den Hardwaresicherheitsspielräumen wäre und ein Fehler bringt oft das gesamte System herunter;

e: Wenn das Vorliegen eines Versagens während des Tests oder bei der Produktion realisiert wird, ist das Lokalisieren seiner Ursache (Fehler) oft extrem schwierig.

[0007] Dies liegt daran, weil die Software fundamental anders als die Hardware darin ist, dass sie nicht den gleichen vereinfachten Regeln der Nähe von Ursache und Wirkung gehorcht. Das heißt, der Softwarefehler liegt nicht irgendwo nahe an seiner Manifestation (Versagen), sondern kann fast überall sonst im System lokalisiert sein.
[0008] Ein Industriedurchschnitt ist 5 bis 10 Fehler pro 1000 Zeilen Code für Systeme, welche die Prüfung passiert haben und in Produktion sind. Aus Gründen der Schwierigkeiten des Lokalisierens von Fehlern nimmt man viele Fehler hin. Programmierer und ihre Manager trauen sich nicht, den Code anzufassen, da jeglicher Versuch, den Code zu verbessern oder zu modifizieren, statt dessen neue Fehler einführen und das System weiter brechen könnte.
[0009] Das folgende Zitat aus dem Jahre 1988 ist auch 1993 noch gültig:
[0010] "Im Prinzip könnte es so funktionieren. Die Defektrate für sorgfältig hergestellte Programme könnte etwa 5 Fehler pro 1000 Zeilen sein, so dass ein 1 Mio.-Zeilenprogramm nur 5000 Probleme haben müsste. Selbst unter der Annahme, dass ein eindeutiger Testpunkt benötigt wird, um jeden darzustellen, sind tausende von Versuchen nicht unmöglich. Der Trugschluss kann offensichtlich sein, unterstreicht jedoch mehr unsere Intuition über Tests, als wir zugeben wollen: Die Punkte können wenige sein, aber wir wissen nicht, wie wir sie finden können. Schlimmer noch, wir kennen keinen semantischen Weg zum Suchen, keinen Weg, ausgewählte Punkte zu beurteilen, und keinen Weg, um zu bestimmen, wann man aufhören soll. Wenn also 5000 Punkte so untersucht worden sind und keine Fehler gefunden worden sind, wissen wir nicht, ob es keine gibt, oder ob 10 zurückbleiben oder 5000". ... Vom technischen Standpunkt ist das, was Testen am meisten erfordert eine stabile grundlegende Theorie ..."
"Special section on software testing"
Communications of the ACM, Juni 1988
[0011] Das Software "Re-engineering" wurde ein Schlagwort der 1990er. IBM System Journal, Band 28, Nr. 2, 1989 hat den Titel: "Programm understanding: Challenge for 1990's".
[0012] Gemäß der IBM-Studie wurden 30 Milliarden Dollar weltweit für Softwarewartung aufgewendet und

diese Zahl wuchs mit der Ansammlung neuer Softwaresysteme und dem "Altern" der alten Software.

[0013] Gemäß der MIT-Studie werden für jeden einem neuen Softwareprojekt zugewiesenen Dollar 9 Dollar für diese Projektwartung ausgegeben.

[0014] Das Softwaremagazin, Seite 95 vom November 1990 erwähnt eine Studie von Price Waterhouse, die eine Existenz eines 100 Milliarden-Dollar-pro-Jahr Markts für Systemarbeit zum Korrigieren von Fehlern an existierenden Systemen evaluiert hat.

[0015] Die heutige Technik des Software-Debuggings basiert auf der "Steuerungsunterbrechungs" Technik. So arbeitet sie. Die Analyse wird von der Quelle durch Einstellen von "Steuerungsunterbrechungen" (Control breaks) durch eines der folgenden Verfahren aktiviert:

a: Der Programmierer studiert den Quell-Code und setzt manuell die Steuerungsunterbrechung durch Positionieren des Cursors oder einer anderen Zeigevorrichtung auf der spezifizierten Quellcodezeile; oder

b: Der Programmierer befiehlt einem Debugging-Werkzeug, den Code zu aktivieren (verfolgen), indem es ihn in einer Geschwindigkeit interpretiert, die niedrig genug ist, um in der Lage zu sein visuell das Ausführen zu verfolgen, indem automatisch die derzeit aktive Zeile des Quell-Codes hervorgehoben wird. In diesem Modus ist eine Unterbrechung nach jeder ausgeführten Quellanweisung verfügbar; oder

c: Es wird angewiesen, dass eine Steuerungsunterbrechung an jedem Befehl eines spezifischen Typs gesetzt wird, beispielsweise nach jedem Eingabestatement oder vor jeder Aufrufen (Durchführen) Anweisung;

[0016] Der Zweck einer Steuerungsunterbrechung besteht darin, vorübergehend einen Ausführungsprozess anzuhalten, um manuell den Wert von Programmvariablen zu untersuchen.

[0017] Die folgenden Annahmen werden gemacht, wenn die "Steuerungsunterbrechungs" Technik verwendet wird:

(I) Ein Programmierer setzt Steuerungsunterbrechungen an relevanten Quellcodestellen;

(II) Unter Annahme von (I) weiß ein Programmierer, welche Variablen untersucht werden sollen;

(III) Unter Annahme von (I) und (II) trifft ein Programmierer die korrekte Entscheidung darüber, was die Ursache eines Versagens ist;

[0018] Die Wahrscheinlichkeit korrekter manueller Analyse ist daher gleich dem Produkt der Wahrscheinlichkeiten von (I), (II) und (III). Beispielsweise gegeben, dass die Wahrscheinlichkeit korrekter Entscheidungen an jedem dieser Schritte 10% ist, dann wäre die Wahrscheinlichkeit eines korrekten Ergebnisses der manuellen Analyse 0,1%.

[0019] Wie wir sehen, ist die traditionelle "Steuerungsunterbrechungs"-Technik des Versuchens, die Ursache eines Versagens zu lokalisieren, in keinster Weise automatisiert und führt alle Möglichkeiten menschlichen Fehlers während der kognitiven Entscheidungen ein:

– Wo prüfen?;
– Was prüfen?;
– Was bedeutet es?

ein.

[0020] Einige Arbeit ist im Bereich des automatischen Feststellens, welche Quellcodeanweisungen während der Testprozessausführung abgedeckt worden sind und welche nicht, gemacht worden.

[0021] Das US Patent 4,853,851 an Horsch offenbart ein "System zum Feststellen der Codeabdeckung eines getesteten Programms basierend auf statistischen und dynamischen Analyseaufzeichnungen", das das Verfahren des Messens des Verhältnisses zwischen der Anzahl der in einem Computerprogramm während seines Testens ausgeführten Instruktionen und der Gesamtanzahl von Instruktionen in diesem Programm ist.

[0022] Obwohl allgemein gilt, dass, je mehr Anweisungen während des Tests abgedeckt werden, um so besser, stellt die Tatsache, dass ein Prozessschritt ausgeführt worden ist, keine Informationen über seine Gültigkeit bereit:

– Die bestimmte ausgeführte Anweisung kann von vornherein falsch konstruiert worden sein,
– die bestimmte Anweisung könnte an einem falschen Platz oder zur falschen Zeit ausgeführt worden sein;
– zusätzlich könnte die bestimmte Anweisung eine Anweisung gewesen sein, die das Prozessergebnis nicht verändert oder zu ihm nicht beiträgt;

[0023] US Patent 4,802,116 an Ward et al. offenbart einen "programmierten Controller", der einen Maschinenprozess steuert und sein Debuggen ermöglicht. Hier sind die Anwendungsprogramme spezifisch dafür ausgelegt, spezifische Steuerungsschemata für Abschnitte der Maschine zu simulieren, indem sie als eine oder mehr getrennte Schleifen organisiert sind. Der eingebaute Steuerungsmechanismus zeichnet die Zustände dieser Schleifen zusammen mit den für die Übergänge zwischen solchen Zuständen verantwortlichen Bedingungen auf.

[0024] Diese Technologie ist auf Anwendung nur bei solchen Prozessen beschränkt, die speziell anhand sehr

spezifischer Syntaxregeln codiert sind, um aus unabhängigen Schleifen zu bestehen, wobei die Mittel zum Verfolgen der Zustände dieser Schleifen in den, den Prozess steuernden Programmen vorcodiert sind. Die Aufzeichnung der Prozessgeschichte wird durch Aufzeichnen eines jeden Blocks von Anweisungen und jeder, verschiedene Blöcke verbindende logische Bedingungsanweisung repräsentierende Sequenz vorgenommen. Die Analyse der Systemfehler ist nicht bis zum Grade automatisiert, den die eX-Maschine bereitstellt, da sie nicht durch Referenz auf das tatsächliche Prozessergebnis (wie im Falle der eX-Maschine) durchgeführt wird, sondern durch eine Analyse der aufgezeichneten Sequenz von Programmzuständen und daher nicht ein derartig hohes Niveau der Betrachtung und der Anwenderschnittstelle bereitstellt, wie die eX-Maschine.

[0025] Der Stand der Technik zeigte keine generischen Prozesse, die in der Lage sein würden, automatisch die Position von Logikfehlern in einer Computersoftware zu lokalisieren, selbst mit einem Grad an Unsicherheit innerhalb eines definierten Bereichs. Dies lag am allgemeinen Glauben, dass eine solche Automatisierung nicht möglich ist, aufgrund des Folgenden:

1. Wie zuvor erwähnt, gehorcht die Software, anders als Hardware nicht derselben Regel von Nähe von Ursache und Wirkung und für ein spezifisches Versagen verantwortliche Softwarefehler könnten fast überall innerhalb des Softwaresystems liegen;
2. Ein Softwaresystem weiß nicht, was von ihm als Ergebnis erwartet wird und daher kann es nicht ein Versagen (unerwartetes Ergebnis) wahrnehmen, solange nicht erwartete Ergebnisse in das System eincodiert werden. Die Arbeit wurde gemacht, indem demonstriert wurde, dass das Präcodieren von Programmspezifikationen eine Aufgabe ist, die in ihrer Komplexität und ihrem Fehleranfälligkeitscharakter nicht anders ist als das Programmieren dieser Softwarefunktionalität von vornherein.

[0026] Zusätzlich würde der Versuch, Softwareversagen durch Präcodieren von Systemspezifikationen zu eliminieren, die folgenden Eigenschaften von Softwareversagen übersehen, da ein Softwareversagen nicht absolut definiert werden kann.

– Ein Softwareversagen ist relativ zu veränderten Bedingungen (Spezifikationen) bei Eingabe und Ausgabe;
– ein Softwareversagen ist relativ zu einer veränderten Betriebsumgebung;
– ein Softwareversagen ist relativ zu individuellen Erwartungen an Systemergebnissen, wobei diese Erwartungen nicht explizit dokumentiert sind;

[0027] Wir werden hier hinzufügen, dass das Problem beim Lokalisieren eines Softwarefehlers in der großen Relativität der Korrektheit des elementaren Softwareprozesses (Programmquellanweisung) liegt.

[0028] Der Elementarprozess lebt in drei Dimensionen. Die ersten zwei sind die zwei Dimensionen des Programmalgorithmusraums, der durch Betrachten des Flussdiagramms verstanden werden könnte. Statische Programmanalyse wird in diesem zweidimensionalen Raum durchgeführt. Die dritte Dimension ist eine Zeit innerhalb des Programmausführungsprozesses. Eine dynamische Analyse muss im Raum und in der Zeit arbeiten.

[0029] Der Unterschied zwischen Programmstatik und Programmdynamikanalyse ist so groß wie der Unterschied zwischen klassischer und zeitgenössischer Mechanik:

[0030] Da "niemand jemals einen Platz bemerkt hat, außer zu einer Zeit, oder eine Zeit ... außer an einem Platz" (H. Minkowsky, "Space and Time") hat ein Elementarprozess innerhalb eines Programms, der seine eigene codierte Position innerhalb des Algorithmenraums des Programms hat, eine Bedeutung nur zu einer Zeit, wenn und falls die Programmsteuerung zu ihm kommt, wobei die Bedeutung zu unterschiedlichen Zeiten innerhalb des Prozesses unterschiedlich ist.

[0031] Der Unterschied der Elementarprozessbedeutung ist bedingt durch den potentiellen Unterschied in den Ausführungsgeschichten vor und nach den Ausführungen dieses Elementarprozesses (Anweisung) zu unterschiedlichen Zeiten innerhalb des Prozesses.

[0032] Im Stand der Technik fehlten die folgenden Haupttechnologieschritte, die nötig sind, um in der Lage zu sein, das Lokalisieren von logischen Fehlern in Computersoftware zu automatisieren und allgemein die dynamische Analyse des Programms zu verbessern:

I) Eine neue Theorie zum Automatisieren von Programmanalyse, welche die Attribute ihrer Relativität einführt und automatische Logikanalyse machbar macht;

II) basierend auf der erwähnten Theorie eine Technologie, die aus Prozessen besteht, die anhand des Wissens über "Korrektheit" und "Unsicherheit" von Zielprozesselementen arbeiten, die einen Mechanismus zum automatischen Minimieren dieser Unsicherheit aufweist, wobei die Prozesse das Ansammeln und Verteilen des erwähnten Wissens zwischen Zielprozesselementen gestatten, womit automatisch die "Intelligenz" des Zielprozessmodells vergrößert wird;

In späteren Beschreibungen werden wir dien Begriff "Wissen" verwenden, um das Verständnis von "Korrektheit" oder ihrer "Unsicherheit" eines Zielprozesselements zu bestimmten Momenten innerhalb des Zielprozesses zu beschreiben.

III) basierend auf der erwähnten Theorie eine Technologie, welche eine Schnittstelle auf hohem Niveau zwischen einem Anwender und dem Zielsoftwaresystem gestattet, welche dem analytischen Werkzeug gestatten würde, am relevanten Platz und Zeit in Bezug auf das relevante Ziel oder die Analyse automatisch initiiert zu werden, d.h. in Bezug auf die relevanten Programmanweisungen und das relevante Systemverhalten;

[0033] (Automatische Analyse kann nicht auf der kognitiven Herausforderung aufgebaut werden, zu versuchen, zu verstehen, wo die Steuerungsunterbrechungen gesetzt werden sollen).
[0034] Wir stellen uns die erwähnten technologischen Schritte als integralen Bestandteil zukünftiger Betriebssysteme vor. Analog zur statischen Syntaxanalyse der Compiler wird ein solches Betriebssystem dynamische Analyse des Zielsystemverhaltens bis zu dem Grad bereitstellen, bei dem es in der Lage ist, mit wachsendem Niveau an Sicherheit, die Position des logischen Fehlers zu lokalisieren.
[0035] Das Analog der unterschiedlichen Grade von Unsicherheit ist ebenfalls bei der Compilerdiagnose vorhanden: Der Compiler detektiert nicht, dass anstelle eines Zeichens "a" ein Zeichen "b" vorliegen sollte; statt dessen berichtet er, dass das Problem eines spezifischen Typs innerhalb eines spezifischen Bereichs einer Anweisung oder einer Gruppe von Anweisungen vorliegt.
[0036] Ohne eine solche partiell unsichere Analyse des Compilers würden wir nicht in der Lage sein, Syntaxfehler irgendwelchen relativ komplexen Codes zu behandeln.
[0037] Dieselbe Automatisierung war lange für den Bereich der Logikfehleranalyse fällig. Die eX-Maschine bietet diese Art von Automatisierung einem weit komplizierteren und vor der vorliegenden Erfindung nicht automatisierten Bereich – Lokalisierung von Position und Typ von logischen Fehlern.
[0038] Die vorstehend erwähnten technologischen Schritte müssen auf der Basis eines sehr generischen, aber sehr stringenten Modellrepräsentierungsprozessalgorithmus gebaut werden, wobei das Modell aus existierenden Softwaresystemen extrahiert werden konnte.
[0039] In der folgenden Beschreibung des Begriffs eX-Maschine werden wir die Kombination der folgenden Prozesse verstehen, die weiter beschrieben werden:

– "Pseudoparallel" (d. h. "Verzögert sequenzielle" oder "verzögert parallele") synchronisierte Ausführung des eX-Modells und des Zielprozesses innerhalb der traditionellen Projektcodeimplementierung;
oder
Ausführung des eX-Modells als eine Komponente des Zielprozessobjektcodes innerhalb der eX-Objektcodeimplementierung;
– das Aufbauen von Systemeingabe-und-Systemausgabeprozesssynchronisierten Karten;
– das Laufenlassen der eX-Analyse-Maschine, die:
– Ein Wissen über Ursache-Effekt-Beziehungen und ihre Korrektheit für die Ausführungsereignisse der Zielprozesselemente erzeugt;
– dieses Wissen zwischen den Zielprozesselementen verbreitet;
– unter Verwendung des akkumulierten Wissens über den Zielprozess, den Bereich der Fehlerposition oder den Bereich, der als ein Ergebnis von veränderten Systemerwartungen modifiziert werden muss, definiert;

[0040] Die Essenz der eX-Maschinen-Technologievorteile kann kurz wie folgt beschrieben werden:
[0041] Um den Fehler in einem Programm zu lokalisieren oder allgemein das Programm zu verstehen, muss ein Programmierer traditionell maßgebliche Detektivarbeit durchführen. Zusätzlich dazu, dass sie menschlichen Fehlern unterworfen ist und daher ein Potential zum Erzeugen sekundärer Fehler aufgrund der Fehler in der manuellen Analyse aufweist, kann diese manuelle Programmanalyse oft zu kompliziert sein, um durchgeführt zu werden.
[0042] Unsere Erfindung führt die Detektivarbeit durch und automatisiert einen großen Teil der manuellen Analyse. Diese Analyse wird hier mit einer Präzision durchgeführt, die mit der Anhäufung von Wissen über das Zielprozessverhalten wächst. Der Kern der Erfindung liegt darin, in der Lage zu sein, dieses Wissen zu akkumulieren und zu verwenden. Die Erfindung studiert die Ereignisse und ihre Effekte innerhalb des Zielprozesses.
[0043] Falls ein Ereignis – etwas, das tatsächliche Existenz aufweist – als ein Effekt verstanden wird, der durch andere Effekte erzeugt wird – eine Bedingung oder ein Auftreten, das zu einer Ursache rückverfolgbar ist, dann können wir sagen, dass unsere Erfindung durch Studieren der Effekte der Elementarprozessereignisse arbeitet, d. h. Ursache-Wirkungs-Beziehung innerhalb des Zielprozesses.
[0044] Anstelle der Analyse des Zielprozesses in der traditionellen Weise, d. h. Vorwärtsgehen (Verfolgen des Zielprozesses) erzeugt die Erfindung die Analyse (automatisch) durch Rückwärtsgehen von den Wirkungen aus.
[0045] Dieser Ansatz ist genau das, was der menschliche Geist gewohnt ist, d. h. wir sehen uns die Wirkungen an, die uns umgeben, und versuchen, ein Verständnis über die diese Wirkungen verursachenden internen Prozesses zu gewinnen, d. h. versuchen, ein Ursache-Wirkungsverständnis durch Zurückgehen von einer Wir-

kung zu einer Ursache aufzubauen.

[0046] Die Tatsache, dass das Computerprogramm durch einen Menschen gebaut worden ist, sollte nicht den Weg ändern, in dem wir es verstehen müssenbeginnend von der Analyse der Wirkungen, nicht durch einfaches Verfolgen der Elementarschritte in Vorwärtsrichtung.

[0047] Da die Zielprozessergebnisse die finalen Wirkungen des Zielprozesses sind, gestattet das In-der-Lage-Sein, die finalen Wirkungen zu ihren Ursachen zurück zu verfolgen, eine Originalschnittstelle, wo die Programmanalyse von der Programmausgabe ausgeht.

[0048] Innerhalb unserer Analyse "rechtfertigt eine erwartete resultierende Wirkung die Kette von Zwischenwirkungen" oder "rechtfertigt ein erwartetes resultierendes Ereignis die Kette der Zwischenereignisse".

[0049] Dies gestattet nichtoptimale Kalkulationen, deren Definition sowieso relativ ist, aber die grundlegenden Prinzipien für Berechnungsanerkennung (Wirkungsanerkennung) ergibt und daher die grundlegenden Prinzipien für Vorwärtsprogrammentwicklung auf der Basis der existierenden Prozesse.

[0050] Die eX-Maschine studiert die Effekte innerhalb des Zielprogramms hinter der Bühne durch die innerhalb der eX-Analyse-Maschine erzeugten Prozesse.

[0051] Der Begriff "Analysemaschine" ist durch Charles Babbage vor ungefähr 150 Jahren unter Bezug auf das erste fundamentale Konzept des mechanischen Computers, an dem er gearbeitet hat, erfunden worden. Seine Idee des Ersetzens eines menschlichen Computers durch einen mechanischen Computer ergab sich aus seiner Besessenheit von durch menschliche Computer erzeugten Fehlern.

[0052] Diese Erfindung der eX-Maschine wurde durch die Fehler bei der menschlichen Programmierung für derzeitige Computer katalysiert.

[0053] Die eX-Analysemaschine lernt durch Akkumulieren der Erfahrung des Zielprozessverhaltens, was das korrekte Verhalten war, was falsches Verhalten war, welches Verhalten nicht analysiert wurde und welches Verhalten fehlerhaft analysiert wurde.

[0054] Dieser Prozess kann mit dem kognitiven Lernprozess des menschlichen Geistes verglichen werden, der Assoziationen aufbaut und unbewusste Reflexe aufbaut.

[0055] Innerhalb des kognitiven Lernens nehmen die Schritte, die uns zu den korrekten Ergebnissen geführt haben, einen speziellen Status in unseren Lernprozess ein. Im Fall von erzeugten späteren unerwarteten Ergebnissen werden sie zuletzt bezweifelt. Fall die Analyse uns zwingt, diese zuvor aufgebauten Reflexe oder Assoziationen zurückzunehmen, ist dies normalerweise ein etwas schmerzhafter Vorgang, der uns zwingt, die vorherigen Ergebnisse zu reanalysieren, welche die Reflexe oder Assoziationen zuerst gebaut haben.

[0056] Innerhalb der eX-Analysemaschine wird dieses Verlernen durch "Verlieren des Wissens" repräsentiert und tritt auf, wenn ein Versagen im Systemergebnis realisiert wird, das zuvor als korrekt verstanden worden ist.

[0057] Der eX-Maschinenlernprozess wird durch die Zielprozessvorwärtsausführung verbreitet und wird durch Akkumulieren der Systemergebnisse, die als korrekt (erwartet) akzeptiert werden, aktiviert. Daher sind die Ergebnisse der Analyse vom menschlichen Faktor abhängig, jedoch werden menschliche Aktivität und damit menschliche Fehler hier auf die Analyse der Endergebnisse – sichtbarer Ergebnisse – beschränkt.

[0058] Diese Flexibilität ist tatsächlich notwendig, um verschiedene Analyseergebnisse für veränderte Systemspezifikationen zu gestatten und die Vorwärtsprogrammentwicklung auf Basis von existierenden Prozessen zu gestatten. Die eX-Analysemaschine gestattet es, einen Teil der vorherigen Analyse zurückzunehmen, wenn ein zuvor übersehenes Versagen gefunden wird oder die Systemerwartungen (Spezifikationen) verändert werden.

[0059] Eines der ursprünglichen Merkmale der eX-Analysemaschine liegt im Minimieren des Bereichs von Unsicherheit über die mögliche Fehlerposition bei der Berechnung des Programmergebnisses mit dem Anwachsen der Zielprozesszeit, die durch das Akkumulieren der Ergebnisse (Effekte) des Programms sichtbar ist.

[0060] Das Maß der Analysepräzision hängt von der Position des ersten wahrgenommenen Versagens oder der Änderung der Spezifikationen innerhalb der Koordinate der ennrähnten Prozesszeit ab, die durch die akkumulierten Systemergebnisse gemessen wird.

[0061] Gemäß einem Aspekt der vorliegenden Erfindung wird eine Vorrichtung zur automatischen Analyse eines Zielprozesses bereitgestellt, wobei die Vorrichtung den Ort von im Zielprozess enthaltenen Fehlern oder den Ort von Modifikationen innerhalb des Zielprozesses, die aufgrund einer veränderten Ergebniserwartung notwendig sind, bestimmt, wobei die Vorrichtung einen sekundären Prozess des Akkumulierens von Wissen zur Zielprozessfunktionalität durch Wissensinduktion erzeugt, umfassend:

Ein Analyseberechnungsbasisattribut (ACB) für jedes ausführbare Element des Zielprozesses;

Mittel zum Ausführen des Zielprozesses und des sekundären Prozesses,

Mittel für den sekundären Prozess, um ein elementares Wissen zur Zielprozessfunktionalität durch Wissensinduktion aufzubauen, die das elementare Wissen durch einen Platz und eine Zeit innerhalb des Zielprozesses verbreiten,

wobei das elementare Wissen ein Wissen oder eine Unsicherheit über die Korrektheit eines Effekts des Zielprozesselementausführungsereignisses anzeigt, und

wobei das elementare Wissen innerhalb des ACBs aufgezeichnet wird und wobei der Ort des für ein spezifisches Versagen verantwortlichen Fehlers oder der Ort der Modifikation innerhalb des aufgrund von veränderter Ergebniserwartung benötigten Zielprozesses durch Wissensdeduktion erhalten wird, die das durch die Wissensinduktion erhaltene elementare Wissen verwendet, um den Ort als einen fortwährend minimierten Bereich von Unsicherheit innerhalb des Ereignisdefinitionsraums des Ergebnisses zu definieren und die Wissensdeduktion denselben Regeln der Wissensinduktion des Verfolgens von Ereignissen in der zur Zielprozessausführung entgegengesetzten Richtung folgt.

[0062] Gemäß einer anderen Aufgabe der vorliegenden Erfindung wird ein Verfahren zur automatischen Analyse eines Zielprozesses bereitgestellt, umfassend

das Feststellen des Fehlerorts, die im Zielprozess enthalten sind oder eines Orts von Modifikationen innerhalb des Zielprozesses, die aufgrund von veränderter Ergebniserwartung notwendig sind;

Erzeugen eines sekundären Prozesses des Akkumulierens von Wissen zur Zielprozessfunktionalität durch Wissensinduktion;

Etablieren eines Analyseberechnungsbasisattributs (ACB) für jedes ausführbare Element des Zielprozesses;

Ausführen des Zielprozesses und des sekundären Prozesses;

Ausführen des sekundären Prozesses, der ein elementares Wissen zur Zielprozessfunktionalität durch Wissensinduktion aufbaut und verbreitet, der das elementare Wissen durch einen Platz und eine Zeit des Zielprozesses verbreitet, wobei das elementare Wissen ein Wissen oder eine Unsicherheit über die Korrektheit eines Effekts des Zielprozesselementausführungsereignisses anzeigt;

Aufzeichnen des elementaren Wissens innerhalb des ACB; und

Erhalten des für ein spezifisches Versagen verantwortlichen Fehlerorts oder des Modifikationsorts innerhalb des aufgrund veränderter Ergebniserwartung benötigten Zielprozesses durch Wissensdeduktion, die das durch die Wissensinduktion erhaltene Wissen verwendet, um den Ort als einen fortführend minimierten Bereich von Unsicherheit innerhalb des Ereignisdefinitionsraums des Ereignisses zu definieren und die Wissensdeduktion denselben Regeln der Wissensinduktion des Verfolgens von Ereignissen in der zur Zielprozessausführung entgegengesetzten Richtung folgt.

[0063] Die vorliegende Erfindung kann daher ein Verfahren zum Minimieren von Unsicherheiten über die Position eines Fehlers oder eines Orts für notwendige Modifikation innerhalb des Zielprozesses bereitstellen, wo der Fehler die Ursache eines Versagens (unerwarteten Ergebnisses) ist und nicht durch einen Compiler gefunden wird, und wo eine Modifikation benötigt wird, um neuen Ergebniserwartungen zu genügen.

[0064] Dies wird durch Synthetisieren von Prozessen erreicht, die analog zum kognitiven Verstehen der Zielprozessfunktionalität sind.

[0065] Zuerst wird das Modell des Zielprozesses auf Basis seiner statischen Analyse durch die sehr strikten aber allgemeinen Regeln aufgebaut, um eine Basis für die weitere Akkumulation des Zielprozesswissens während seiner dynamischen Analyse zu erzeugen. In diesem Moment hat jede relevante Programmanweisung ihre eigene spezifische Position innerhalb des zweidimensionalen eX-Algorithmenraums.

[0066] Die Striktheit der resultierenden Konstruktion gestattet einen Mechanismus, der durch Aufbauen aufeinanderfolgender "Sichten" automatisch und leicht die Zugänglichkeit (Vorhandensein eines Pfads) zwischen irgendwelchen zwei Elementen des Zielprozesses evaluieren kann. Eine "Zugänglichkeit" definiert "potentielle Abhängigkeit". Nur falls Anweisung (b) von Anweisung (a) zugänglich ist, kann dann die Korrektheit von (a) das Verhalten von (b) betreffen.

[0067] Beispielsweise ist ein wichtiges Ergebnis der automatischen Evaluierung von Elementzugänglichkeit die Möglichkeit, endlich-"nichtterminierbare" Prozesskonstruktionen zu identifizieren – datenunabhängige Endlosschleifen.

[0068] Zur Zeit des Aufbauens des Modells des Zielprozesses stellt die vorliegende Erfindung ein Verfahren zum Extrahierung der Steuerungskomponenten des Zielprozessalgorithmus in eine die Steuerstruktur dieses Prozesses repräsentierende Form ("eX-Algorithmusrahmen") bereit. Der eX-Algorithmusrahmen ist eine strikte Einzeldefinitionsform, die notwendig und hinreichend zum Repräsentieren der Steuerungsstrukturen irgendeines Prozess ist.

[0069] Zweitens wird ein Mechanismus eingeführt, der es gestattet, spezifische dynamische Informationen für jeden relevanten Elementarprozess (Programmanweisung) zu akkumulieren. Diese Information ist für das Verständnis der Elementarprozessfunktionalität relevant und wird an dem Aufbau während der vorgehenden statischen Analyse des Zielprozessmodells angeheftet, wobei sie dieses Modell mit dynamischem Wissen über das Zielprozessverhalten füttert. Wir nennen diese spezifische Information weiterhin ein Elementarprozess-"Wissen". Der erwähnte Mechanismus gestattet es, eine Informations"Ursache-Wirkung"-Beziehung und eine "relative" und "zeitabhängige" Messung der "Korrektheit" von Prozesselementen aufzubauen.

[0070] Die Vorrichtung kann damit ein Mechanismus zum Definieren eines Prozessschrittes als "korrekt" oder "unsicher" innerhalb der in dieser Erfindung definiert Koordinaten von "Prozessraum" und "Prozesszeit" {"Prozessadresse" und "Lokalzeit"} und in dieser Erfindung definiert Messungen der Prozess-"Korrektheit" und "Unsicherheit" bereitstellen. Das erwähnte Elementarprozess-"Wissen" wird kontinuierlich akkumuliert und zwi-

schen verschiedenen Prozesselementen verbreitet. Wenn das Zielprozessversagen oder ein vom erwarteten abweichendes Ergebnis wahrgenommen wird, wird dieses Wissen verwendet, um die Position der notwendigen Modifikation innerhalb des Zielprozesses zu identifizieren.

[0071] Auch werden Mittel zum Schützen "korrekter" Elemente des Zielprozesses vor Veränderung während dieser Zielprozesswartung durch Aufbauen einer Wissensbasis der "korrekten" Funktionalität des Prozesses offenbart.

[0072] Auch wird ein Mechanismus für eine Schnittstelle auf hohem Niveau zwischen dem Anwender und dem akkumulierten Wissen des Zielprozesses durch die eX-Maschine offenbart.

[0073] Eine Multimediapräsentation des Zielprozessverhaltens kann erzeugt werden, um seine Diagnose zu vereinfachen.

[0074] Auch wird eine Form des Zielprozessobjektcodes – eX-Objektcodes und das Verfahren seiner Ausführung offenbart. eX-Objektcode ist zur Programmanalyse, sowohl statisch als auch dynamisch, und zur Programmwartung geeigneter. Es vereinfacht und in einigen Fällen eliminiert traditionelles Recompilieren und Neulinken des Zielprozesscodes, die normalerweise durchgeführt werden müssen, um den Prozess zu modifizieren.

[0075] Diese Merkmale werden erreicht durch Extrahieren, aus dem Quellcode des Zielprozesses einen für diese Zielprozessstatikinformation spezifischen, der zum Aufbauen eines Modells, welches den Zielprozessalgorithmus repräsentiert, notwendig ist; auf der Basis dieses Modells Aufbauen eines dynamischen Wissens des Zielprozessverhaltens; Einführen des Mechanismus zum Aufbauen eines Verständnisses der Ursache-Wirkungs-Beziehungen innerhalb des Zielprozesses; Einführen des Mechanismus zum Aufbauen "zeitabhängiger" und "relativ" "Korrekt-unsicher" Attribute von Elementarprozessen innerhalb des Zielprozesses; und Einführen der Verfahren zum Verbreiten dieses "Korrekt-unsicher"-Wissens zwischen Zielprozesselementen während dieser Zielprozessausführung und daher automatisches Anheben des Intelligenzniveaus des Zielprozessmodells.

[0076] Diese Erfindung bietet auch einige Merkmale an, die besser unter Bezugnahme auf die Detailbeschreibung unten verstanden werden.

Kurze Beschreibung der Zeichnungen

[0077] **Fig.** 1 ist ein Flussdiagramm des beispielhaften Zielprozesses;

[0078] **Fig.** 2 ist ein eX-Algorithmusrahmen des durch **Fig.** 1 dargestellten beispielhaften Zielprozesses;

[0079] **Fig.** 3 demonstriert, wie komplexe Logikbedingungen zu zwei oder mehr reduzierten Logikbedingungen reduziert werden;

[0080] **Fig.** 4 repräsentiert das reduzierte Flussdiagramm des in **Fig.** 1 dargestellten beispielhaften Zielprozesses;

[0081] **Fig.** 5(A) ist ein eX-Graph des durch die **Fig.** 1 dargestellten beispielhaften Zielprozesses und 5(B) repräsentiert die den entsprechenden Elementen des eX-Graph (5A) zugewiesenen Binäradressen;

[0082] **Fig.** 6 demonstriert, wie IF-THEN-ELSE-Konstruktionen im eX-Rahmen und eX-Graph repräsentiert werden;

[0083] **Fig.** 7 demonstriert die Schritte des Aufbauens von eX-Modell, eX-Verwahrungsort und X-Quellcode;

[0084] **Fig.** 8 demonstriert die Hauptkomponenten des eX-Maschinenbetriebs;

[0085] **Fig.** 9 demonstriert den Mechanismus des Definierens von Ausgangszutrittsindikatoren und des Definierens der Zugänglichkeit zwischen irgend welchen zwei Elementen des Zielprozesses durch Aufbauen von aufeinanderfolgenden "Sichten" am Beispiel eines anderen beispielhaften Prozessflussdiagramms;

[0086] **Fig.** 10 demonstriert den Mechanismus des Definierens des "propagierten Sichtspunkts" (PVP) eines Prozessendes (Terminals).

[0087] **Fig.** 11 ist ein Diagramm, welches unterschiedliche Quellen von Fehlern zeigt, die möglicherweise während der Zielprozessimplementierung eingeführt werden;

[0088] **Fig.** 12 ist ein drei Arten von Zielprozessversagen zeigendes Diagramm;

[0089] **Fig.** 13 ist ein Diagramm, das erläutert, wie die eX-Maschine den Bereich der Fehlerpositionsunsicherheit während der Zielprozessorwärtsausführung minimiert;

[0090] **Fig.** 14 demonstriert, wie Ereignisdefinitionsraum aufgebaut wird;

[0091] **Fig.** 15 demonstriert, wie Korrekt-Algorithmusraum verbreitet wird;

[0092] **Fig.** 16 demonstriert, wie das "Wissen" um eine Prozesselementkorrektheit innerhalb einer spezifischen Prozessadresse (PA) akkumuliert wird;

[0093] **Fig.** 17 demonstriert den Mechanismus des Definierens der "Vorläufer"-Ereignisse;

[0094] **Fig.** 18(A) demonstriert den Prozess der Ausführung des eX-Objektcodes; 18(B) demonstriert zwei Komponenten des eX-Objektcodes;

[0095] 18(C) demonstriert den Prozess des Interpretierens des eX-Verwahrungsorts, wie er durch das eX-Modell gesteuert wird.

[0096] **Fig.** 19 demonstriert die Implementierung der ACB (Analyzed Calculation Base, analysierte Kalkulationsbasis)-Konstruktion.
[0097] **Fig.** 20 eX-Graph (**Fig.** 20A) und eX-Verwahrungsort (**Fig.** 20B) eines in
[0098] **Fig.** 21A dargestellten beispielhaften Prozesses.
[0099] **Fig.** 21 beispielhafter Prozess (**Fig.** 21A) und sein X-Quell-Code (**Fig.** 21B).

Beschreibung der bevorzugten Ausführungsform

[0100] Die Beschreibung wird mit ihren in der vorliegenden Reihenfolge eingeführten Hauptschritten präsentiert werden:

1 Aufbauen des eX-Rahmens des Zielprozessalgorithmus;
2 Aufbauen des eX-Graphen und des Reduzierten Flussdiagramms des Zielprozesses;
3 Definitionen von eX-Maschinenprinzipien des Evaluierens der Zielprozess-"Korrektheit" und Lokalisieren von "Unsicherheit" innerhalb des Zielprozesses als ein Ort von Modifikationen aufgrund eines Fehlers oder veränderter Systemanforderungen;
4 Definition der eX-Maschinenprinzipien der Zielprozesswissensverbreitung (Wissensinduktion);
5 Aufbauen des eX-Modells des Zielprozesses;
Definition der analysierten Kalkulationsbasis (ACB);
Aufbauen des eX-Verwahrungsortes des Zielprozesses;
6 Bauen des X-Quellcodes;
Minimale Überbauinstrumentierung zur Erfassung der Zielprozessverhaltenshistorie;
Aufbauen der Anwender-eX-Maschinen-Zielprozessschnittstelle vermittels einer inkrementierten systemausgabeprozesssynchronisierten (ISOPS) Karte;
Bauen der systemeingabeprozesssynchronisierten (SIPS) Karte;
7 Beispiel für Zielprozess, sein eX-Modell, eX-Verwahrungsort und X-Quellcode;
8 "Pseudoparallele" synchronisierte Ausführung des Zielprozessmodells:
"Verzögert sequenzielle" und "verzögert parallele" eX-Modell-Ausführung;
9 Ablaufen der eX-Analysemaschine. Aufbauen und Verbreiten der analysierten Berechnungsbasis von Ursache-Wirkungs-Beziehungen innerhalb des Zielprozesses;
Wissensinduktion;
10 Aktivieren der eX-Analysemaschine-"Wissensdeduktions"-Prozesse unter Bezugnahme auf die ISOPS-Karte, um den Bereich von Fehlerpositionen oder Orten für Modifikationen mit konstant abnehmender Unsicherheit zu lokalisieren;
11 interpretieren von eX-Graph und eX-Verwahrungsort;
12 eX-Objektcode und seine Ausführung;
Ausführung des eX-Modells als ein Teil des Zielprozessobjektcodes;
13 Recompilation und Neu-Linken des eX-Objektcodes;
14 Implementieren von eX-Maschinenprozessen auf allen Stufen von Test und Produktion;
15 resultatorientierte Programmierung (ROP);
16 Implementieren eines Programmstethoskops;

1 Aufbauen des eX-Rahmens des Zielprozessalgorithmus:

[0101] In der folgenden Beschreibung werden wir zeigen, wie das eX-Modell aus dem eX-Graphen aufgebaut wird. Das Konstruieren des eX-Graphen des Zielprozesses kann vorhergehen einem, begleitet sein von oder gefolgt sein von dem Aufbauen des eX-Rahmens dieses Prozessalgorithmus.
[0102] Obwohl der Schritt des Aufbauens des Algorithmusrahmens komplett umgangen werden kann, werden wir diesen Schritt beschreiben, um in bequemer Weise einige von dem Rahmen und dem eX-Graphen geteilte wichtige Attribute zu demonstrieren. Zusätzlich kann der eX-Algorithmusrahmen jederzeit als eine sehr kompakte und strikte (Einzeldefinition) Repräsentation der Zielprozesssteuerstruktur verwendet werden.
[0103] Die folgende Beschreibung enthält verschiedene Definitionen und Axiome. Diese sind notwendig, da einige der Objekte und Begriffe, mit denen die eX-Maschine arbeitet, nicht zuvor definiert worden sind.
[0104] Das Folgende ist ein Satz von Objekten, mit denen der eX-Rahmen, eX-Graph und das eX-Modell arbeiten. Wir werden auf diesen Satz als eX-Objektesatz Bezug nehmen.
{[,A,X,L,D,E,W,N,=,R,*,C,+,S,/,!,(,),V,i}
wobei
"[" Prozesseintritt,
"A" arithmetische oder Zuweisungsanordnung
"X" Funktion oder Subroutinenaufrufanweisung, außer Grafikbibliothekssubroutine oder Funktion Beispiel: CALL, PERFORM;

"L" IF-Anweisung (eine Eingabe und zwei Ausgaben);
"D" Schleifenanweisung Beispiel: DO, FOR, WHILE;
"E" Eingabeanweisung Beispiel: READ;
"W" Ausgabeanweisung oder Graphikbibliotheksubroutine oder Funktion; Beispiel: WRITE(), CALL MOVE-TO(x,y,xy), wobei {x,y}-Bildschirmkoordinaten
"N" Nullanweisung Beispiel: CONTINUE;
"=" STOP-Anweisung
"R" RETURN-Anweisung;
"*" GO TO-Anweisung;
"C" Schleifenendanweisung oder ihre Position, falls angenommen, Beispiel: CONTINUE, ENDDO;
"+" zugewiesene GO TO-Anweisung;
"S" ASSIGN (ALTER)-Anweisung;
"/" Eintritt in GO TO, oder seine Position, Beispiel: Position eines Bezeichners oder Absatznamens, die von irgend einem GO TO referenziert werden;
"!" ENDIF oder seine Position, falls angenommen;
"(" Öffnung des LC-Felds;
")" Schließen des LC-Felds;
"V" ELSE;
"i" – Index des Prozesselements innerhalb seiner Objektklasse für eX-Objekte außer "("und")";
– Niveau des LC-Felds für eX-Objekte "("und")";

[0105] Bei der folgenden Beschreibung wenden wir uns einem Computerprogrammprozess zu, das ein einzelnes Modul ist, obwohl unsere Ergebnisse einfach auf einen, ein System von Programmen repräsentierenden Prozess ausgedehnt werden können.

[0106] Wir verstehen das Modul entweder als ein Hauptprogramm oder ein einzelnes Unterprogramm (Unterroutine, Funktion oder Absatz) mit einem Eintrittspunkt. Obwohl das folgende Beispiel ein Programm mit einem Ausgang repräsentiert, wird die beschriebene Implementierung in derselben Weise für Module mit mehreren Ausgängen funktionieren. Die Implementation der eX-Maschine für Module mit mehreren Eintritten kann aus der hier beschriebenen Implementierung der eX-Maschine für ein Einzeleintrittsmodul erhalten werden. **Fig.** 2 repräsentiert den eX-Rahmen des durch sein Flussdiagramm in **Fig.** 1 illustrierten Beispielzielprozesses.

Definition F1: "Reduzierte Logikbedingung" (LC)

– ist ein Prozesselement mit einem Steuerungseingang und zwei Steuerungsausgängen, welche die zukünftige Richtung des Prozesses durch ihre Boolsche (0 oder 1) Lösung bestimmen.

[0107] Im eX-Objektesatz ist die LC durch "L" oder "D"-Objekte repräsentiert; wir werden die negative Lösung 0-Lösung oder Niederpotentiallösung und die positive Lösung die 1-Lösung oder Hochpotentiallösung nennen.

[0108] Wenn der Zielprozess Steuerungselemente mit mehr als zwei Ausgängen beinhaltet, werden solche Steuerungselemente während ihrer Repräsentierung im eX-Rahmen und eX-Graphen automatisch auf zwei oder mehr LCs reduziert. Man beziehe sich auf **Fig.** 3. Die Anzahl der resultierenden LCs ist gleich der Anzahl von Steuerungsausgängen minus eins, da jede LC die Anzahl von Steuerungsausgängen um eins erhöht.

Definition F2: "Hauptzweig"

– ist ein, einen Pfad durch den Prozess in Richtung seiner Ausführung repräsentierender Zweig, vom Ursprung des Hauptzweigs zum "Prozessende", wenn alle erfüllten Logikbedingungen durch ihre Niederpotentiallösung passiert werden.

[0109] Es gibt zwei Arten von Ursprüngen der Hauptzweige:
1: Prozesseintritt (in **Fig.** 1 und **Fig.** 4 als START dargestellt), der den "Eintrittshauptzweig" startet;
2: Hochpotentiallösung von LCs.

Definition F3: "Prozessenden"

– sind die folgenden Objekte des eX-Objektsatzes:

	eX-Rahmenrepräsentation
GO TO	"*" "+"
Schleifenenden	"C"
RETURN	"R"
STOP	"="

[0110] Es gibt zwei Arten von Prozessenden:

Definition F4: "Ausgangsenden"

– sind Enden {"R", "="}, die zu einem Verlassen des Prozesses führen.

Definition F5: "Rücksprungenden"

– sind die Enden {"*", "+", "C"}, die dem Prozess zu dem Element zurückführen, das bereits im selben Hauptzweig oder in dem Hauptzweig definiert ist, der im eX-Graph links von dem derzeitigen Hauptzweig definiert ist.

[0111] Daher hängt die Länge des Hauptzweigs von der Position seines Endes ab, das wiederum von der Anordnung der Hauptzweigdefinitionen abhängt.
[0112] Die Anordnung der Darstellung der Hauptzweige im eX-Rahmen und eX-Graph kann durch das Prinzip des "Nächstniederen Potentials" beschrieben werden.

Die "Nächstniederpotential"-Ordnung des Parsens des Algorithmus

Schritt 1:	Position an START.
Schritt 2:	Führe Parsen des Prozesses durch "Niederpotentiallösungen" (0-Lösungen) erfüllter LCs durch, bis es uns zu dem zuvor passierten Element bringen würde oder bis zu einem EXIT.
Schritt 3:	Zurück bis zu der LC, die zuletzt durch ihr "Niedriges Potential" gegangen ist, falls vorhanden; Positioniere an ihrer "Hochpotentiallösung"; Setze fort ab Schritt 2; Falls alle LCs bereits durch ihre "Hochpotential"-Lösung gegangen sind, ist das Parsen des Zielprozessalgorithmus abgeschlossen

[0113] Jede Ausführung von Schritt 1 baut einen Hauptzweig auf, der durch ein "Ausgangs-Ende" oder "Rücksprungende" beendet wird. Daher werden jene GO TO-Konstruktionen, die vorwärts verzweigen, aus dem eX-Rahmen und eX-Graphen eliminiert.
[0114] Axiom: Der Prozess mit n reduzierten LCs enthält (n + 1) Enden und (n + 1) Hauptzweige, wobei jeder Hauptzweig hinreichend durch sein Ende definiert ist.
[0115] Axiom: Die Gesamtheit von (n + 1) ist notwendig und hinreichend zum Repräsentieren aller Elemente des Prozesses.
[0116] Erläuterung: Beim Prozess ohne LCs, d. h. ohne Bedingungssteuerung, gibt es nur einen Zweig – den Eintrittshauptzweig. Jede LC wird die Anzahl von Hauptzweigen um eines erhöhen, durch Eröffnen eines neuen Zweigs durch ihre Hochpotentiallösung.
[0117] Jeder Hauptzweig und sein Ende sind im eX-Graph durch ihre b-Koordianate (**Fig.** 5(A)) definiert. Der Rahmen des Prozessalgorithmus wird auf Basis des folgenden Boolschen Ausdrucks aufgebaut: LA V !L B, der sich liest: Falls L, führe A aus, sonst (Nicht-L) führe B aus. Im eX-Rahmen führen wir Klammern: '(',')' ein, welche den Bereich von Logikniveaus und Enden repräsentieren, die Kreise: {'*', '+', 'C'} und Ausgänge {'R', '='} ermöglichen. Auch vermeiden wir '!L', da es offensichtlich ist, dass nach dem ELSE-Zeichen 'V', das zu der spezifischen L gehört, 'nicht L' folgen wird. Somit erhalten wir die Form: (iLAVB)i, wo A und B entsprechend "0-Unterfeld" und "1-Unterfeld" von LC L sind und i das "Niveau des LC-Felds" repräsentiert; die Definitionen sind folgende:

Definition F6: "t-Durchlauf"

– ist ein Durchlauf vom Prozesseintritt bis zum Prozessende;

[0118] Axiom: In jedem Prozess gibt es (n + 1) t-Durchläufe, wobei n die Anzahl von LCs im Prozess ist. Jeder

t-Durchlauf ist eindeutig durch sein Ende (**Fig.** 5(A)) identifiziert.

Definition F7: "Zweig der LC"

– ist der Teil des t-Durchgangs, der durch die LC durchgeht, welcher der LC folgt.

[0119] Jede LC hat einen oder mehrere 0-Zweige und einen oder mehrere 1-Zweige. In **Fig.** 1 weist LC L1 2 1-Zweige auf, die durch "d" und "h" markiert sind.
[0120] Definition F8: "0-Unterfeld der LC" – ist eine Kombination aller Prozesselemente, die auf 0-Zweigen dieser LC positioniert sind.

Definition F9: "1-Unterfeld der LC"

– ist eine Kombination aller Prozesselemente, die auf 1-Zweigen dieser LC positioniert sind.

Definition F10: "Feld der LC"

– wird aus 0-Unterfeldern und 1-Unterfeldern dieser LC kombiniert.

Definition F11: "Niveau des LC-Felds" (Logikniveau)

– ist die Abfolge dieser LC im t-Durchgang, der durch sie durch geht.

[0121] Beispiel: in **Fig.** 1 sind die folgenden Logikniveaus vorhanden:
L1 öffnet Logikniveau **1**
L2 und L3 öffnen Logikniveau **2**
D1 öffnet Logikniveau **3**
[0122] In **Fig.** 1 sind die Hauptzweige durch die Linien "a", "b", "c", "d", "e" markiert. Sie sind durch die Enden identifiziert, die in **Fig.** 5(A) an den b-Koordinaten gefunden werden können, entsprechend 1, 2, 3, 4, 5. Das 0-Feld von LC L1 ist in **Fig.** 1 durch die gestrichelte Linie "f' gezeigt und das 1-Feld von LC L1 ist durch die gestrichelte Linie "g" gezeigt.
[0123] Der eX-Rahmen wird durch Repräsentieren der Hauptzweige in einer speziellen Anordnung gemeinsam mit den durch ihre entsprechenden Niveaus präsentierten LCs repräsentiert. Man beziehe sich auf **Fig.** 2: eX-Rahmen ist in der Form eines algebraischen Ausdrucks aufgebaut, der mehrere Niveaus an Klammern enthält:

(. .(.. (. .)..). .(. .).. .)

1 2 3 3 2 2 2 1

[0124] Die Regel eines algebraischen Ausdrucks ist, dass die äußeren (niedrigeres Niveau) Klammern nicht geschlossen werden können, bevor die Inneren (höheres Niveau) Klammern geschlossen sind.
[0125] Im eX-Rahmen repräsentieren Klammern logische Niveaus. Dieselbe Regel gilt: Dass das untere (äußere) Niveau an Logikfeld nicht geschlossen werden kann, bis das Logikfeld höheren (inneren) Niveaus geschlossen ist.

Definition F12: "Feld der LC wird geöffnet"

– wenn wir anfangen, dieses Feld im Rahmen zu repräsentieren.

Definition F13: "Feld der LC wird geschlossen"

– wenn es komplett im Rahmen repräsentiert ist.

Definition F14: "Aktives Feld der LC"

– ist das höchste noch offene Feld von LC.

Definition F15: "Prozesssegment"

– ist der Teil des Hauptzweigs, der zwischen den folgenden Prozesselementen positioniert ist: Prozessein-

tritt, Prozessenden und LCs.

[0126] Prozessenden und LCs beenden Prozesssegmente. Sie gehören zu den Segmenten, die sie beenden. Ein Ende eines Prozesssegments seiend gehören jedem LC gleichzeitig zwei andere Segmente: Ein negatives Segment der LC (durch ihre 0-Lösung) und ein positives Segment der LC (durch ihre 1-Lösung).
[0127] Segmente besitzen zwei Attribute:
1: Logisches Niveau und
2: Binäradresse (wird weiter beschrieben)

Definition F16: "Logisches Niveau des Prozesssegments"

– ist gleich dem logischen Niveau des LC, zu dem das Segment gehört.

Definition F17: "Eintrittssegment"

– ist das Prozesssegment, das dem Prozesseintritt gehört.

[0128] Das logische Niveau des Eintrittssegments ist 0. Es gibt (2n + 1) Prozesssegmente, wobei n die Anzahl der LCs in dem Prozess ist. Das heißt zwei Segmente für jede LC und ein Eintrittssegment. Beispielsweise: In **Fig.** 1 enthält der Zielprozess 4 LCs und 9 Prozesssegmente:
[0129] Die Bezeichnung der Prozesselemente für den eX-Rahmen und den eX-Graphen des Zielprozesses wird in der Reihenfolge des Aufbaus von eX-Rahmen oder eX-Graph vorgenommen.
[0130] Bei dem in **Fig.** 1 dargestellten beispielhaften Zielprozess:

Segmente:	durch den eX-Objektsatz repräsentierte Prozesselemente:
1:	A1/1 A2 L1
2:	/2 A3 L2
3:	*1
4:	A4 X1 D1
5:	A5 C1
6:	A6 W1 *2
7:	W2 E1 L3
8:	A7=
9:	A8 *1

[0131] **Fig.** 2 ist der eX-Rahmen des Prozesses, repräsentiert durch sein Flussdiagramm in **Fig.** 1. Gemäß der früher definierten nächstniederen Potentialanordnung beginnen wir das Aufbauen des Rahmens des Algorithmus vom Eintrittshauptzweig aus.
[0132] Wenn er durch irgend eines der folgenden Objekte: {*,+,C,R,=} beendet wird, gehen wir zurück zur letzten LC, die durch ihre Niederpotentiallösung (NEIN) gegangen ist und beginnen den Hauptzweig zu durchlaufen, der zur Hochpotentiallösung (JA) dieser LC gehört. Auf diese Weise können wir konsequenterweise alle (n + 1)-Hauptzweige des Prozesses repräsentieren, wobei n die Anzahl von LCs innerhalb des Prozesses ist.
[0133] **Fig.** 6 demonstriert die Darstellung einer IF-THEN-ELSE-Konstruktion im eX-Rahmen und eX-Graphen. Das 1-Feld von L1 (P3) wird erst repräsentiert, nachdem das 0-Feld L1 (P2, P4) geschlossen wird (d. h. vollständig repräsentiert).

Verfahren zum Extrahieren eines Algorithmus des Zielprozesses in den eX-Algorithmusrahmen, der eine generische Einzeldefinition-Form ist, die durch die "Nächstniedere Potential"-Regel aufgebaut ist;

Schritt 1: Weise dem aktiven Logikniveau 0 zu; Positioniere auf dem Prozesseintritt;

Schritt 2: Laufe durch den Hauptzweig; jedem durchlaufenden ausführbaren Element wird ein Name zugewiesen, der aus einem eX-Objektsatzelement mit entsprechendem Index im Inkrement von 1 besteht;
(Beispiel: A1, A2, L1, A3, L2, *1 repräsentiert den Eintrittshauptzweig); die Anweisung wird dann unter dem zugewiesenen Namen im eX-Prozessverwahrungsort (Fig. 7 und Fig. 8) gespeichert; wenn die LC im derzeit durchlaufenen Hauptzweig erfüllt wird, wird sie durch ihre Niederpotentiallösung (NEIN) durchlaufen;

Diese LC öffnet das nächste Logikfeld, wobei das aktive Logikniveau um 1 erhöht wird;
Jede durchlaufene LC wird im Rahmen durch die öffnenden Klammern gemeinsam mit einer Ganzzahl repräsentiert, welche das Niveau des geöffneten Logikfelds repräsentiert, gefolgt von dem Zeichen L oder D mit dem entsprechenden Index: (i Lj oder (i Dj;

Schritt 3: Wenn ein Prozessende erreicht wird, wird das Segment beendet.
Falls das Ende ein Rücksprungende {(*,+} ist, wird ein Eintritt /i an der Position der Steuerungsrückkehr mit einem Index (i) im Inkrement von 1 eingefügt, es sei denn, es existiert bereit ein Eintrag;
Dem Rückkehrende wird ein Eintrittsindex (i) zugewiesen.

Schritt 4: Überprüfe aktives Logikniveau:
IF ein aktives Logikniveau 0 ist, dann
eX-Rahmenkonstruktion ist abgeschlossen;
STOP;
ELSE
das Unterfeld des aktiven Logikniveaus wird als geschlossen annonciert;
ENDIF

Schritt 5: Untersuche geschlossenes Unterfeld;
IF das geschlossene Unterfeld das 0-Unterfeld war, dann
das Zeichen 'V' (ELSE) wird im Rahmen platziert;
das Parsen wird vom Hochpotential (1-Lösung) der entsprechenden LC aus wieder aufgenommen, um den Hauptzweig dieser LC aufzubauen;
1-Unterfeld dieses Niveaus wird als geöffnet annonciert;
GO TO-Schritt 2:

ELSE (das geschlossene Subfeld war das 1-Subfeld)

Annonciere aktives Logikfeld als geschlossen, da sowohl sein 0-Subfeld als auch sein 1-Subfeld geschlossen sind;

das Schließen des Logikfelds wird im Rahmen durch Schließen der Klammern zusammen mit der Ganzzahl beschrieben, welche das Niveau des geschlossenen Logikfelds bezeichnet: ')i';

das aktive Logikniveau wird um 1 gesenkt;

das gerade geschlossene Logikfeld (i....)i wird als das Ende für das Segment angesehen, das die entsprechende LC beendet;

wenn das Feld der LC geschlossen ist, wird das entsprechende Segment als beendet annonciert;

GO TO Schritt 5;

ENDIF

[0134] Das vorstehende Beispiel demonstriert, dass bei der eX-Rahmenkonstruktion (**Fig.** 2) die Felder der LCs der höheren Logikniveaus innerhalb der Felder der LCs des niedrigeren Logikniveaus liegen.
[0135] In jedem Prozess kann es ein Maximum von F Logikfeldern des Niveaus N geben, wo F = 2 hoch (N – 1).

2 Aufbauen des eX-Graphen und des Reduzierten Flussdiagrams des Zielprozesses;

[0136] Das Folgende erläutert den Prozess des Aufbauens des eX-Graphen (**Fig.** 5(A)) und des Reduzierten Flussdiagramms (**Fig.** 4) des Zielprozesses (**Fig.** 1).

Definition G1: "Reduziertes Flussdiagramm" (**Fig.** 4)

– ist eine Darstellung des eX Graphen in der Form eines Flussdiagramms.

[0137] Es wird anhand derselben Koordinaten k und b und durch dieselben Prinzipien wie der eX-Graph aufgebaut. Der folgende Prozess, der die Regeln zum Aufbauen des eX-Graphen beschreibt, gilt auch für das Aufbauen des reduzierten Flussdiagramms.
[0138] Es wird ein "reduziertes" Flussdiagramm genannt, da alle möglichen Änderungen in der Richtung der Steuerung auf eine vermindert sind – nach rechts, d. h. der Hochpotentiallösung einer LC. Die Niederpotentiallösung ändert die Richtung der Steuerung nicht dadurch, dass der Steuerung gestattet wird, weiter herunter zu "fallen". Die Sprünge werden nur rückwärts gestattet (hoch und links) und sind durch die {Ende – sein Rücksprungeintritt} Referenzpaare repräsentiert, beispielsweise: {*1/1} oder {*2/2} oder {C1 D1} in **Fig.** 5(A).

Definition G2: Negativer Zustand von LC (0-Zustand)

– ist der Zustand, wenn, falls evaluiert, LC zu ihrer 0-Lösung führen würde.

[0139] Der Zustand von LC ist durch den Zustand von Variablen definiert, die bei seiner Evaluierung beteiligt sind.

Definition G3: Positiver Zustand von LC (1-Zustand)

– ist der Zustand, wenn, falls evaluiert, LC zu ihrer 1-Lösung führen würde.

Definition G4: "Binäradresse" (BA)

– ist die Kombination von Einsen und Nullen, welche die Lösungen von LCs im t-Durchlauf hinauf bis zu dem Prozesselement sind, bei dem dem Prozesseintritt die Binäradresse von 1 zugewiesen ist.

[0140] BAs der Prozesselemente, die zum selben Segment gehören, sind dieselben. BA vom 0-Segment der LC wird aus der BA der LC berechnet, in dem sie mit 2 multipliziert wird (d. h., 0 wird rechts platziert).
[0141] BA von 1-Segment von LC wird aus der BA dieser LC berechnet, indem sie mit 2 multipliziert wird und 1 addiert wird (d. h., 1 wird rechts platziert).
[0142] Beispiel: in **Fig.** 4 ist die BA von L3 11, die BA von A7 ist 110 und die BA von A8 ist 111.
[0143] Der eX-Graph wird nur in zwei Richtungen aufgebaut. In unserer Implementation wird er vertikal herunter und horizontal zur Rechten aufgebaut.
[0144] Die Hauptzweige sind vertikal nach unten gerichtet und repräsentieren Bahnen, an denen sich die Ausführung bewegt, bis die Bahn durch den positiven Zustand einer der LCs, die längs des Weges positioniert sind, geändert wird.
[0145] Jede Bahn wird auf ihren eigenen b-Koordinaten aufgebaut und die Bahnen kreuzen sich niemals.
[0146] LCs haben durch ihre Hochpotentiallösung die Fähigkeit der Steuerungsumschaltung auf die andere Bahn, deren b-Koordinate mit dem positiven Zustand dieser LC assoziiert ist und rechts von der derzeitigen Bahn positioniert ist. Dieses Umschalten ist im eX-Graphen durch die horizontale Linie nach rechts repräsentiert.
[0147] Der eX-Graph wird durch dieselben Prinzipien entweder aus dem eX-Rahmen oder durch Parsen des Zielprozesses aufgebaut. Es wird auf zwei Koordinaten aufgebaut: K und B, die von (1,1) bis (MK, NB) wachsen, wobei MK das maximale K und NB das maximale B ist, das auch die Anzahl der Hauptzweige ist.
[0148] Die Bequemlichkeit des Startens vom eX-Rahmen besteht darin, dass der Satz der Hauptzweige bereits definiert ist als solche Teile des Rahmens, die durch ein "V"-Zeichen getrennt sind. Die Hauptzweige sind im eX-Rahmen bereits in der richtigen Reihenfolge dargestellt.

Verfahren zum Extrahieren eines Algorithmus des Zielprozesses in einem "doppelgerichteten beendeten" eX-Graphen durch die "Nächstniedere Potentialregel";

Unter Bezug auf Fig. 5(A):

Schritt 1: **Der eX-Graph wird an dem Prozesseintritt gestartet, dem die (k,b) Koordinaten von (1,1) gegeben sind; Dem Prozesseintritt wird die Binäradresse 1 zugewiesen; Der Eintrittshauptzweig wird gestartet;**

Schritt 2: **Beim Durchlaufen des Hauptzweigs wird jedes Element auf dem eX-Graphen mit Inkrement der k-Koordinate und konstanter b-Koordinate repräsentiert; jedem Element wird eine BA des Segments zugewiesen, zu dem es gehört;**

außer wenn der eX-Graph aus dem eX-Rahmen aufgebaut wird, müssen die folgenden (a) und (b) durchgeführt werden:

(a) Das Prozesselement wird dem aus dem entsprechenden Element des eX-Objektsatzes und Index im Inkrement von 1 bestehenden Namen zugewiesen;

(b) die Anweisung wird im Prozessverwahrungsort unter den zugewiesenen Namen aufgezeichnet (Fig. 7);

Schritt 3: Wenn LC beim Durchlaufen durch den Hauptzweig erfüllt wird, wird seine 0-Lösung durchlaufen;

Schritt 4: Wenn das Konstruieren des Hauptzweigs durch ein Prozessende gestoppt wird:

IF alle LCs bereits ihre 1-Lösung durchlaufen haben,
THEN eX-Graph wird aufgebaut.
STOP
ELSE
Rückverfolge den derzeitigen t-Durchlauf gegen die Richtung der Prozessausführung, bis eine LC getroffen wird, die ihre 0-Lösung durchlaufen hat;
starte den Hauptzweig dieser LC an den eX-Graph-Koordinaten (next_k,next_b),
wobei next_k=(k der LC)+1 und
next_b die nächste verfügbare Bahn (Koordinate b) ist
GO TO Schritt 2;
ENDIF

[0149] Es ist wichtig anzumerken, dass eine der zwei eX-Graphdimensionen, NB, exakt das ist, was als eine der Programmkomplexitätsmessungen akzeptiert wird. McCabes Theorie "A Complexity Measure" wurde in

IEEE Transactions On Software Engineering, Band SE-2, Nr. 4, Dezember 1976, Seiten 308–320, veröffentlicht.

[0150] "Zyklomatische Komplexität" wird von McCabe als maximale Anzahl der linear unabhängigen Pfade definiert und diese sind für einen einzelnen Eintritt und Ausgang definiert durch den Rahmen: V(G) = e – n + 2
wobei V(G) – zyklomatische Komplexität
e – Anzahl von Kanten
n – Anzahl von Knoten (Eintrittsknoten + Austrittsknoten + Verzweigungsknoten)

[0151] Wir werden diesen Rahmen als McCabes Rahmen der zyklomatischen Komplexität bezeichnen.

[0152] Eine Kante ist unser Segment und die Zahl von Kanten e = 2q + 1, wobei q die Anzahl der LCs ist, d. h. unsere Anzahl von Prozesssegmenten;

[0153] Ein Verzweigungsknoten ist unsere LC und die Anzahl der Verzweigungsknoten ist q, wobei q die Anzahl der LCs ist;

[0154] Dann gilt für einen einzelnen Eintritts- und einzelnen Austrittsknoten gemäß McCabes Rahmen zyklomatischen Komplexität:

$$e = (2q + 1)$$

$$n = 1 + 1 + q = (q + 2)$$

$$V(G) = (2q + 1) – (q + 2) + 2 = q + 1$$

V(G) = q + 1, wobei q die Anzahl von LCs ist

[0155] Somit ist die durch McCabe definierte "Zyklomatische Komplexität" gleich der Anzahl unserer LCs plus 1 und gleich unserer NB, wobei NB die Anzahl der Hauptzweige ist.

[0156] Der vorige Rahmen korrespondiert mit dem Ergebnis des in Patent Nr. 4853851, erteilt 1989 Beschriebenen: "Die Zyklomatische Anzahl gleich der Anzahl von linear unabhängigen Pfaden ist die Anzahl von Pfaden, gezählt am breitesten Teil des Programmflussgraphen. Dieser Ansatz der Berechnung der Zyklomatischen Anzahl ist noch nicht von McCabe beschrieben worden".

[0157] Die zweite Dimension des eX-Algorithmusraums (MK) könnte ebenfalls verwendet werden, um eine Algorithmuskomplexität zu beschreiben. MK repräsentiert den längsten Weg vom Prozesseintritt zu einem Ende und daher ist die Zahl der Elementarprozesses der längste t-Pfad. Der Raum des Rechtecks mit den Dimensionen [MK, NB] erscheint als eine gute Anzeige der Zielprozessalgorithmuskomplexität. Wir nennen diesen Raum einen "Algorithmusraum" des Zielprozesses.

[0158] Beispielsweise ist der Algorithmusraum des Beispiels von **Fig.** 1 [14 × 5].

[0159] **Fig.** 5(A) demonstriert folgende nützliche Eigenschaften des eX-Graphen.

1. Jedes Element kann eindeutig durch zwei Koordinaten (k,b) adressiert werden;
2. Der Graph wird nur in zwei Richtungen aufgebaut – in unserer Implementierung herunter und zur Rechten – wobei das Herunterfallen nur durch Verschieben nach rechts durch die Hochpotentiallösung einer LC unterbrochen werden kann.
3. Prozessenden, die nicht "Ausgangs"-Enden (STOP oder RETURN) sind, können den Prozess "rückspulen". Diese Rückspulung ist nur rückwärts gestattet, d. h. hoch links. Dies ist so, da die Vorwärtssprünge während der Konstruktion des eX-Rahmens und eX-Graphens durch die Definition des Hauptzweigs eliminiert worden sind.
4. Jedem Segment wird die Binäradresse zugewiesen, welche die Logikposition dieses Segments innerhalb des Zielprozesses eindeutig identifiziert.
5. Der eX-Graph gestattet für eine einfache Feststellung der "Zugänglichkeit" zwischen Prozesselementen.

Definition G5: "Element s2 ist von Element s1 zugänglich"

– wenn es einen Pfad von s1 nach s2 gibt.

[0160] Das heißt, s2 ist von s1 zugänglich, falls es möglich ist, dass eine gewisse Eingangsdatenkombination s2 einige Zeit nach der Ausführung von s1 ausgeführt werden kann.

[0161] Die Existenz von "Zugänglichkeit" zwischen zwei Prozesselementen ist eine notwendige Bedingung, damit die Korrektheit eines Elements das Ergebnis eines anderen Prozesselements beeinflusst, d. h., dass "Zugänglichkeit" die notwendige Bedingung für die "Ursache-Wirkungs"-Beziehung zwischen zwei Prozesselementen ist.

[0162] Kürzlich sind verschiedene Definitionen von Programmabhängigkeiten vorgeschlagen worden, um die Effekte innerhalb der Programme zu studieren. Zwei grundlegende Typen von Programmabhängigkeiten sind: "Steuerungsabhängigkeiten" und "Datenflussabhängigkeiten". Programmabhängigkeiten werden für solche

kritischen Zwecke wie etwa Softwaretesten, Debuggen und Wartung verwendet. Bis jüngstens sind die meisten Verwendungen von Programmabhängigkeiten nur informal gerechtfertigt worden, falls überhaupt im Hinblick auf die Schwierigkeiten der Implementierung einer solchen Verwendung.

[0163] In der Arbeit von A. Podgurski und L. A. Clark, "A Formal Model of Program Dependencies and Its Implications for Software Testing, Debugging, and Maintenance," IEEE Transactions of Software Engineering, 16,9, September 1990, Seiten 965–979 präsentieren die Autoren den Begriff "semantische Abhängigkeit" als ein gewisses Konzept, das eine notwendige Bedingung für eine Anweisung ist, um potentiell das Ausführungsverhalten (und damit das Ergebnis) einer anderen Anweisung zu beeinflussen. Jedoch beweist diese Arbeit, dass verschiedene Generalisierungen von Steuerungs- und Datenflussabhängigkeiten keine hinreichenden Informationen sind, um automatisch Programmfehler zu identifizieren, da sie nicht hinreichend sind, auch nur die Existenz von "semantischer Abhängigkeit" zu identifizieren, was wiederum jedenfalls nicht hinreichend währe, um Programmfehler zu identifizieren.

[0164] Die in dieser Erfindung präsentierte Methode zur automatischen Bestimmung von Elementzugänglichkeit könnte im Hinblick auf die "semantische" Abhängigkeit als ein praktisches Verfahren zum Evaluieren der notwendigen Bedingung für die Existenz der "semantischen" Abhängigkeit zwischen jeglichen zwei Elementen des Zielprozesses definiert werden.

[0165] Anders ausgedrückt evaluiert das dargestellte Verfahren praktisch das Potential für:

1. Dass Anweisung s1 für das Missverhalten von s2 verantwortlich ist;
2. dass Anweisung s2 durch die Modifizierung von s1 beeinflusst wird;

[0166] In der Lage zu sein, die Zugänglichkeit zwischen zwei Prozesselementen zu bestimmen, löst auch die Frage der Anwesenheit von "endlichen, nicht beendbaren" datenunabhängigen Endlosschleifen. Wir werden im Beispiel von **Fig.** 9 zeigen, wie das Verwenden der eX-Graph-Binäradressenarithmetik leicht die Existenz definieren kann von:

- Zugänglichkeit zwischen Si und Sj;
- Teile des Prozesses, die datenunabhängige Endlosschleifen erzeugen;

[0167] Es wird allgemein akzeptiert, dass die Frage, ob das Programm enden wird oder nicht, als unlösbar bewiesen ist. Wir können diese Frage der Prozess"-Beendbarkeit", d. h. der Fähigkeit zu enden, in zwei Kategorien unterteilen:

– Datenunabhängige Endlosschleifen
– datenabhängige Endlosschleifen

[0168] Datenunabhängige Endlosschleifen bestehen aus Prozesselementen, die keinen Zugriff auf "Ausgangs"-Enden haben.

[0169] Eine Datenabhängige Endlosschleife ist eine solche Konstruktion, dass nach ihrem Betreten die Steuerung niemals einen Ausgang erreichen wird (STOP oder RETURN), aus einem anderen Grund als der Abwesenheit von Steuerdurchgängen zu diesen Ausgangsenden. Der Grund ist in diesem Fall entweder:

a: Die Abwesenheit der Manipulation von Programmvariablen, die potentiell den Steuerungsdurchlauf auf den spezifischen Steuerungsdurchläufen ändern könnte oder

b: Änderung dieser Programmvariablen in der Richtung, welche die Werte der LCs weiter vom Ändern ihrer Zustände zwischen Niederpotentialzustand und Hochpotentialzustand wegführt.

[0170] In beiden Fällen a: und b: fällt die Steuerung in eine Endlosschleife, die durch dieselben Endungen in derselben Abfolge hindurchgeht, ohne jemals in der Lage zu sein, ein "Ausgangs"-Ende zu erreichen.

Definition G6: *bn

– bezieht sich auf das Ende mit der b-Koordinate gleich n.

[0171] (Beispiel: in **Fig.** 10 bezieht sich *b3 auf das GO TO-Ende auf b = 3 und bezieht sich *b5 auf das "Ausgangs"-Ende auf b = 5)

Definition G7: MBn

– bezieht sich auf einen auf der b-Koordinate gleich n aufgebauten Hauptzweig.

[0172] (Beispiel: MB3 in **Fig.** 9 bezieht sich auf den auf Koordinate b = 3 aufgebauten Hauptzweig;

[0173] Beispiel einer Datenabhängigen Endlosschleife: Falls es keine Elemente am Durchgang zwischen /1 und *1 gibt, welche die durch L2 und L3 examinierten Werte redefinieren können, produziert Ende *b1 eine

datenabhängige Endlosschleife, sobald L2 und L3 zum ersten Mal von ihren Niederpotentiallösungen gelöst werden (da der Zustand von L2 und L3 sich während der Vorwärtsausführung des Zielprozesses niemals ändern wird.)

[0174] Beispiel für eine datenunabhängige Endlosschleife: Datenunabhängige Endlosschleifenkonstruktion ist in **Fig.** 9 durch die Hauptzweige MB2, MB3, MB4 dargestellt. Falls L3 jemals positiv gelöst wird, geht der Prozess in die Endlosschleife, unabhängig davon, welche Datenmanipulationsanweisungen innerhalb dieser Zweige kodiert sind.

[0175] Während Konstruktionen von datenabhängigen Endlosschleifen ein Potential für eine Prozess-"Nichtendbarkeit" bergen, sind sie nicht notwendigerweise fehlerhafte Konstruktionen, da es so sein könnte, dass Eingangsdaten niemals derart sein können, dass sie die LCs in den Zustand der endlosen Schleife bringen würden.

[0176] Ein Eintrag /q in **Fig.** 9 entspricht keinem Ende und wird nur als ein Beispiel der Tatsache gezeigt, dass die Position von L3 im negativen Unterfeld von L2, wobei das Rückspulende *b1 die Prozesssteuerung direkt vor L2 zurückbringt, nicht notwendigerweise ein Fehler ist, selbst falls L2 und L3 nicht datenabhängig auf dem Pfad /1 nach *1 sind. Das ist so, weil, falls L3 die Steuerung bis zum Eintrag /q empfangen hat, das Rückführen der Steuerung zum Eintrag /1 nicht notwendigerweise dazu führt, dass L2 negativ gelöst wird, wodurch eine Endlosschleife erzeugt würde.

[0177] Datenunabhängige Endlosschleifen sind andererseits definitive Fehler bei der Prozesskonstruktion. Die Detektion von datenunabhängigen Endlosschleifen wird durch Evaluieren einer Zugänglichkeit zwischen "Rückspul"-Enden und "Ausgangs"-Enden durchgeführt. Derselbe Prozess würde die Evaluierung der Zugänglichkeit zwischen jeglichen zwei Elementen des eX-Graphen gestatten (beispielsweise zwischen Si und Sj in **Fig.** 9).

[0178] Der eX-Graph gestattet die Erklärung der Zugänglichkeit zwischen seinen Elementen, wie durch die folgenden drei Potentiale bestimmt:

– "Frei-Fallpotential" innerhalb eines Hauptzweigs;
– "Schiebepotential" von LCs;
- "Rückspulpotential" eines Endes;

[0179] "Freier Fall"-Potential wird durch die Natur der sequenziellen Prozeduren bestimmt, wenn ein Prozesselement nach dem anderen aktiv wird.

Definition G8: "Schiebepotential von LC"

– ist das Potential von LC, die b-Koordinate als Ergebnis einer positiven Lösung der LC zu inkrementieren.

[0180] Beispiel: in **Fig.** 5(A) haben die LCs: L2, D1 und L3 ein Schiebepotential von 1 und L1 hat ein Schiebepotential von 3.

Definition G9: "Rückspulpotential des Endes"

– ist das Potential eines Endes, BA zu dekrementieren.

[0181] Der Wert von BA ist wie folgt festgelegt:
Falls bbb eine Bitsequenz ist, dann gilt bbb0 > bbb
bbb1 > bbb0
bbb1 > bbb00

[0182] Das heißt, um sie zu vergleichen, sind BAs linksbündig, entsprechende Bitpositionen werden links nach rechts verglichen bis zur ersten Abweichung, wo 1 > 0 > _, und wobei _ eine leere Position ist und eine Bitposition links hat eine höhere Signifikanz.

[0183] Beispiel in **Fig.** 5:

BA(L2) > BA(L1), da 10 > 1

BA(W2) > BA(A3), da 11 > 10

BA(W2) > BA(A5), da 11 > 1010

Definition G10: "Segmentadresse" (SA)

– ist ein Paar {BA,k}, wobei BA eine Binäradresse des Segments und k eine k-Koordinate innerhalb des

Segments ist.

[0184] Segmentadressen können verglichen werden. Beim Vergleichen von SAs {BA,k} hat BA eine höhere Signifikanz, d. h. (11,7) > (10,7), und (11,7) > (11,6).
[0185] Beispiel **Fig.** 9: Segmentadresse von /3 ist {10010,rk3}

Definition G11: "ElementSicht"

– Element s2 ist innerhalb der Sicht von Element s1 positioniert oder s2 wird von s1 gesehen und s1 sieht s2, falls SA (s2) ein Abkömmling von SA(s1) ist;

Definition G12: SA(s2) ist ein Abkömmling von SA(s1),

falls BA von s2 aus BA von s1 durch Verknüpfen von Binärziffern an der rechten Seite aufgebaut wird oder falls BA(s1) gleich BA(s2) und k(s2) größer ist als k(s1);
s2 ist in Sicht von s1, falls es einen Pfad von s1 nach s2 gibt, der Teil eines t-Durchgangs ist. Man erinnere sich, dass der t-Durchgang keine Enden durchläuft.

[0186] Beispiel: In **Fig.** 4 ist A6 in Sicht von A3, da 10 linksbündig innerhalb von 1010 ist; (A6 wird von A3 gesehen und A3 sieht A6).
[0187] Beispiel: 101 kann auf 1011 linksbündig gemapt werden, daher liegt jegliches Element innerhalb Element **1011**: {A6,W1,*b3} in Sicht jeglichen Elements von Segment **101** {A4,X1,D1}
[0188] Beispiel: X1 liegt in Sicht von A4, da ihre BAs beide **101** sind und k von X1 größer als k von A4 ist.

Definition G13: "Sicht des Rückspulendes

– ist die Sicht seines Rückspuleintritts; die Sicht von "Ausgangs"-Ende ist leer.

Definition G14: "s2 ist von s1 zugreifbar"

– falls s2 innerhalb des "Zugriffsfeldes" von s1 liegt.

[0189] Definition G15: "Zugriffsfeld des Prozesselements" ist zusammengesetzt aus allen Sichten, die es sukzessive besitzt. Der Ausdruck sukzessive bedeutet, dass die Sichten durch die Ende-Sichten verbreitet werden.
[0190] Somit wird ein Zugriffsfeld des Prozesselements kombiniert aus der Sicht des Elements + der Sichten des Endes, das es sieht + der Sichten von Enden, die von diesen Enden gesehen werden usw.)
[0191] Die genauere Definition von "Zugriffsfeld" folgt und wird durch die Definition von "propagierter Sichtpunkt" (PVP) eines in der Definition G22 dargestellten Prozessendes dargestellt.
[0192] **Fig.** 9 stellt 5 für jedes Ende des eX-Graphen aufgebaute Attribute dar. Ein Endattribute sind:
– Die Binäradresse des Endes (TBA);
– Rückspulbinäradresse des Endes (RBA);
– rk des Endes;
– propagierter Sichtpunkt (PVP);
– Ausgangszugriffsindikator (EAI);

Definition G16: Rücksprungbinäradresse (RBA)

– ist die Binäradresse des Rückspulendeintritts; (RBA des Ausgangsendes ist gleich 0)

[0193] rk ist die k-Koordinate eines Rückspulendeintritts; (rk von Ausgangsende ist gleich 0)

Definition G17: "Ausgangszugriffsindikator (EAI)"

– ist ein Indikator, der auf 0 für "geschlossene" Enden und auf 1 für "offene" Enden gesetzt wird.

Definition G18: "Offenes Ende"

– ist ein Ausgangsende oder Rückspulende, das Zugriff hat auf eines der Ausgangsenden.

Definition G19: "Geschlossenes Ende"

– ist ein Rückspulende, das keinen Zugriff auf irgendwelche der Ausgangsenden hat.

Definition G20: "Geschlossener Bereich"

– ist eine Kombination angrenzender Hauptzweige, die durch geschlossene Enden definiert sind.

[0194] Wenn einmal die Steuerung des Prozesses in einen geschlossenen Bereich kommt, würde sie nicht in der Lage sein, herauszukommen, d. h. der Prozess wird nicht enden.
[0195] Beispiel: in **Fig.** 9 definieren {*b2, *b3, *b4} einen geschlossenen Bereich.

Definition G21: Die Position des "Propagierten Sichtpunkts" (PVP) eines Prozessendes:

– wird durch **Fig.** 10 beschrieben.

[0196] **Fig.** 10 zeichnet das Prinzip der Sichtverbreitung des Endes und Einstellen des propagierten Sichtpunkts zweiter oder mehr Prozessenden. In dieser Figur ist eine aufgebaute Sicht durch eine gepunktete Linie repräsentiert und der Durchgang vom Ende zu seinem Rückspuleintritt ist durch eine feste Linie repräsentiert.
[0197] Unter Bezug auf **Fig.** 10(A): Durch unsere Definition ist die Sicht des Endes die Sicht seines Rücksprungeintritts. Falls der Rücksprungeintritt /n von Prozessende *n ein anderes Ende *i sehen kann, dessen Rückspuleintritt /i den Rücksprungeintritt /n sehen kann, sagen wir, dass die Sicht von *n durch die Sicht von *i verbreitet wird und die Sicht von *i der propagierte Sichtpunkt beider Enden *i und *n wird;
[0198] Der propagierte Sichtpunkt eines Endes wird durch die Segmentadresse eines spezifischen Rückspuleintritts repräsentiert.
[0199] Falls die Ende-Sicht nicht durch die Sichten anderer Enden verbreitet wird, ist der propagierte Sichtpunkt dieses Endes seine eigene Sicht, d. h. die Sicht seines Eintritts.
[0200] Derselbe Mechanismus von End-Sichtverbreitung ist gültig zwischen der verbreiteten Sicht verschiedener Enden und der Sicht eines anderen Endes:
[0201] In **Fig.** 10(A) kann der Rückspuleintritt des Endes *k, d. h. /k, das Ende *n sehen, das einen PVP /i aufweist. Daher wird der PVB von *k nach /i verbreitet.
[0202] Beispiel: In **Fig.** 9 ist der PVP von *b1 /1;

PVP von *b2, *b3, *b4 ist /2

PVP von *b5 ist 0, d.h. leer;

PVP von *b6 ist /2;

PVP von *b7 ist /7;

Definition G22: "Das Zugriffsfeld" des Prozesselements

– ist eine Kombination seiner eigenen Sicht + propagierter Sichten aller Enden, die in Sicht dieses Elements positioniert sind.

[0203] Bezugnahme auf **Fig.** 9: Beispiel: Das Zugriffsfeld von Element Si ist die Sicht von Eintritt/1, da sie aus Sichten von Si und PVP von *b1, PVP von *b2, PVP von *b3 und PVP von *b4, die in Sicht von Si sind, zusammengesetzt ist;
PVP von b1 besitzt die Sicht von Si und die Sicht von /2, d. h. den PVP von (*b2, *b3, *b4).
[0204] Beispiel: Das Zugriffsfeld jedes Elements Se, das zu MB2 gehört und die k-Koordinate <= rk2 aufweist, ist die Sicht dieses Elements, da es die Sicht von /2 besitzt, d. h. einen PVP von *b2, *b3, *b4, die in Sicht von Se sind.
[0205] Beispiel: Das Zugriffsfeld von Element L5 ist die Sicht von /2, da es der propagierte Sichtpunkt von *b3 und *b4 ist, die in Sicht von L5 liegen; die Sicht von L5 gehört der Sicht von /2 und wird nicht gezählt;
[0206] Wenn alle Enden innerhalb der Sicht eines Rückspuleintritts diesen Eintritt als ihren PVP reklamieren, ist das Ergebnis der geschlossene Bereich – (datenunabhängige Endlosschleife).
[0207] Beispiel: *b2, *b3 und *b4 beanspruchen /2 als ihren PVP, der wiederum keine anderen Enden in seiner Sicht hat. Dies definiert den geschlossenen Bereich, d. h. datenunabhängige Endlosschleife.
[0208] Um die "geschlossenen" Enden und "geschlossenen" Bereiche zu finden, d. h. die Konstruktionen, welche datenunabhängige Endlosschleifen repräsentieren, bauen wir den Ausgangszugriffindikator (EAI) für jedes Ende des eX-Graphen.

Aufbauen des Ausgangszugriffsindikators (EAI)

[0209] Unter Bezugnahme auf **Fig.** 9: TBA, RBA und rk jedes Endes sind bekannt, nachdem das Konstruieren des eX-Graphen beendet ist.
[0210] Ursprünglich wird der Wert von PVP aus dem Rückspuleintritt des Endes aufgebaut, d. h. rk und RBA des Endes.
[0211] EAI aller Rückspulenden sind ursprünglich mit 0 belegt (d. h. geschlossen gesetzt) und die EAI aller Ausgangsenden sind mit 1 belegt (d. h. offen gesetzt).
[0212] Beim Durchlaufen von b = 1 bis b = NB untersuche SA des PVP des Endes gegen BAs und PVPs aller anderen Enden in einen Versuch, den PVP des derzeit untersuchten Endes zu verbreiten (dekrementieren) und möglicherweise das Ende zu öffnen, falls es noch geschlossen ist.
[0213] Wir werden den PVP von Ende *bi als PVP*bi bezeichnen und werden den EAI von *bi als EAI*bi bezeichnen.
[0214] Bezugnahme auf das Beispiel von **Fig.** 9:

I. Ursprünglich sind die PVPs der Enden auf die SAs ihrer Rücksprungeintritte gesetzt:
BA von PVP*b1 wird auf 10 gesetzt;
BA von PVP*b2 wird auf 1001 gesetzt;
BA von PVP*b3 wird auf 10010 gesetzt;
BA von PVP*b4 wird auf 10011 gesetzt;
BA von PVP*b5 wird auf 0 gesetzt, da *b5 ein "Ausgangs"-Ende ist;
BA von PVP*b6 wird auf 10011 gesetzt;
BA von PVP*b7 wird auf 10 gesetzt;
II. Verbreiten von PVPs und Einstellen von EAIs
(A) Beim Durchlaufen der Enden in der Reihenfolge von *b1 bis *bn, wobei n gleich NB ist, d. h. die Zahl von Hauptzweigen, untersuche jedes Ende in Bezug auf alle anderen Enden des eX-Graphen in einem Versuch, zu:
a: den PVP des Endes durch die früher beschriebene Regel (G21) zu verbreiten.
b: das Ende durch Einstellen seines EAI auf 1 durch die folgende Regel zu öffnen.

Definition G23: Falls der PVP des geschlossenen Endes irgend ein anderes offenes Ende sehen kann, dann wird das geprüfte Ende durch Setzen seines EAI auf 1 als offen annonciert.

[0215] PVP*b1 {10, rk1} sieht *b2, *b3, *b4, *b5, aber keines dieser PVPs der Enden kann PVP*b1 sehen;

PVP*b1 wird nicht geändert.

PVP*b1 {10,rk1} sieht offenes *b5 (1010);

offenes b1: setze EAI*b1 auf 1;

PVP*b2 {1001,rk2} sieht *b3, *b4, die 100110 und 100111 sind, aber keiner von diesen PVPs der Enden, die 10010 und 10011 sind, können den PVP*b2 sehen;

PVP von *b2 ist nicht verändert;

PVP*b2 {1001,rk2} sieht keine offenen Enden, die im

Moment *b1 und *b5 sind, d.h. 1000 und 1010);

EAI*b2 wird auf 0 gehalten;

PVP*b3 {10010,rk3} sieht b2 (10010) und PVP*b2 {1001,rk2} sieht PVP*b3, d. h. {10010,rk3};

setze PVP*b3 auf PVP*b2: {1001,rk2};

das verbreitete PVP*b3 sieht immer noch nicht

irgendwelche geöffneten Enden;

EAI*b3 wird auf 0 gehalten;

PVP*b4 {10011,rk4} sieht *b3 (100110) und PVP*b3 {1001,rk2} kann PVP*b4 {10011,rk4} sehen;

stelle PVP*b4 auf {1001,rk2};

das verbreitete PVP*b4 kann immer noch nicht

irgendwelche offenen Enden sehen;

EAI*b4 wird auf 0 gehalten;

PVP*b5 wird nicht verändert, da er ein Ausgangsende ist;
PVP*b6 {10011,rk6} sieht *b3 und *b4, dessen. derzeitige PVPs {1001,rk2} PVP*b6 sehen können;

setze PVP*b6 auf {1001,rk2};

verteiltes PVP*b6 sieht immer noch nicht irgendwelche

geöffneten Enden;

EAI*b6 wird auf 0 gehalten;

PVP*b7 {10,rk7} kann die Enden von *b1 bis *b6 sehen, aber keiner von diesen kann PVP*b7 sehen;

PVP*b7 wird nicht verändert;

PVP*b7 {10,rk7} kann die offenen Enden (1000 und 1010)

sehen;

öffne *b7: Setze EAI*b7 auf 1;

[0216] In einigen Prozessen (Bezug auf **Fig.** 10(B) wird Ende *i, das durch die Sicht von *n verteilt wird, geprüft, bevor der PVP von *n endlich gesetzt wird, da der PVP von *n wiederum durch die Sicht des anderen Endes aktualisiert werden kann, usw.

[0217] Daher ist der einzige Weg, sicherzustellen, dass alle möglichen Aktualisierungen von PVP und EAI durchgeführt waren, den Schritt (A) des Prüfens aller Enden in der Reihenfolge von *b1 bis *bn noch einmal zu wiederholen. Falls keine Aktualisierungen innerhalb des letzten Durchgangs gemacht worden sind, wird der Prozess der Aktualisierung der PVPs und EAIs beendet.

[0218] Im schlimmsten Fall ist die Anzahl der benötigten Durchgänge durch die Enden, d. h. Ausführungen von Schritt (A), gleich der Anzahl von Hauptzweigen NB, da der Durchgang schließlich den PVP und den EAI von zumindest einem Ende einstellen wird. (In unserem Beispiel wurden alle Aktualisierungen in einem Durchgang durchgeführt und der zweite Durchgang durch *b1 bis *b7 wird keinen der PVP und EAI ändern).

[0219] Im Ergebnis sehen wir: EAIs der Enden *b2, *b3, *b4 und *b6 sind 0, diese Enden sind geschlossen.

[0220] Enden *b2, *b3, *b4 bilden einen geschlossenen Bereich (datenunabhängige Endlosschleife). Ende b6 ist auch geschlossen und sein Rückspuleintritt ist innerhalb des erwähnten geschlossenen Bereichs [*b2–*b4].

[0221] Die gepunktete Linie (a) repräsentiert das Sichtsfeld eines Eintrags /1 {10,rk1}. Dieser Eintrag ist der PVP von *b1. Dies bedeutet, dass irgendein Element des Hauptzweigs b = 1 auf irgend ein Element innerhalb der Sicht von {10,rk1} zugreifen kann.

[0222] Ein Zugriffsfeld von (Si) wird durch die gepunktete Linie (a) repräsentiert.

[0223] Die gepunktete Linie (b) repräsentiert das Sichtfeld eines Eintrags /2 {1001,rk2}. Dieser Eintritt ist PVP von *b2, *b3, *b4 und *b6. Dies bedeutet, dass irgendein Element des Hauptzweigs MB2, MB3, MB4 und MB6 auf irgendein Element innerhalb der Sicht von {1001,rk2} zugreifen kann.

[0224] Durch unsere Definition ist ein Zugriffsfeld eines Prozesselements die Kombination aller Sichten, die es sukzessive besitzt.

[0225] Daher wird ein Zugriffsfeld von Sj aus swn durch die gepunktete Linie (c) und gepunktete Linie (d) repräsentierten Felder kombiniert, da die gepunktete Linie (c) die Sicht von Sj ist und die gepunktete Linie (d) die Sicht von *b6 ist, die von Sj gesehen wird.

[0226] In **Fig.** 9 können wir sehen, dass (Si) auf (Sj) zugreifen kann und (Sj) nicht auf (Si) zugreifen kann.

[0227] Es gibt zwei Schritte im Aufbauen von Zugänglichkeit vom Prozesselement S1 zum Prozesselement S2:

1: Falls SA(s2) ein Abkömmling von SA(s1) (aufgebaut durch Vergleichen ob Binäradressen und BAs gleich sind, dann durch Vergleichen von k-Koordinaten), dann s2 ist zugänglich von s1;

2: Falls der Schritt 1 versagt, dann falls SA(s2) ein Abkömmling von dem PVP irgendeines Endes, das von s1 gesehen wird, dann s2 ist zugänglich von s1;

[0228] Beispiel von **Fig.** 9: Die Attribute (BA, RBA, PVP) prüfend, die für jedes Ende aufgebaut werden, können wir das Folgende feststellen:

a: Si kann Sj nicht sehen, da **101** nicht von **100** gesehen wird; jedoch kann Si auf Sj zugreifen, da eines der Enden, von Si gesehen, *b1, Sj durch seinen PVP {10,rk1} sehen kann, da **101** ein Abkömmling von **10** ist;

b: Sj kann Si nicht sehen, da **100** kein Abkömmling von **101** ist; die Enden, die von Sj gesehen werden, sind die mit BAs **1010** und **1011**; Ende **1010** hat keine(n) RBA oder PVP als Ausgangsende; Ende **1011** hat einen PVP {**1001**,rk1}, welches **100** ebenfalls nicht sehen kann. Daher kann Sj nicht auf Si zugreifen.

3 Definitionen von eX-Maschinenprinzipien des Evaluierens von Zielprozess-"Korrektheit" und Lokalisieren von "Unsicherheit" innerhalb des Zielprozesses als Ersatz für die Modifikation aufgrund eines Fehlers oder veränderter Systemanforderungen;

[0229] **Fig.** 11 stellt die Prinzipien dar, die für jeglichen Computersoftwareprozess gelten. Hier sehen wir die drei Klassen von Prozessfehlern (Fehlern) vom Punkt ihres Ursprungs. Jene drei Klassen enthalten alle möglichen Prozessfehler.

[0230] Die erste Klasse (a) sind während des Prozessdesigns eingeführte Fehler. Ihr Ursprung liegt im Missverstehen der Prozessspezifikationen oder in während des Designs von Schritten, die eine erwartete Ausgabe von einer erwarteten Eingabe erzeugen müssen eingeführten Fehlern. Solche sind Fehler im Prozessalgorithmus.

[0231] Die zweite Klasse (c) sind solche Fehler, die während der Designimplementierung eingeführt worden sind. Dies sind die Fehler, die während des Kodierens des entworfenen Algorithmus in eine vom Compiler verstehbare Programmiersprache eingeführt worden sind. Dies sind "logische" Codierfehler. Man beachte, dass die Syntaxfehler, die schlussendlich vom Compiler lokalisiert werden, hier nicht gezählt werden, da eX-Maschinen "logische" Fehler adressieren, die im Code vorhanden sind, nachdem er vom Compiler überprüft worden ist.

[0232] Während der Ausführung der Zielprozesse während des Testens oder der Produktion könnte es Eingabekombinationen geben, die ungültig sind, aber nicht erwartet worden sind und daher nicht vom Prozessalgorithmus berücksichtigt und abgefangen worden sind. Wir können sagen, dass Fehler dieser Klasse in der Eingabe liegen, d. h. außerhalb des Zielprozesses. Diese sind Eingabefehler (i).

[0233] Der Zweck von **Fig.** 11 ist, zu zeigen, dass alle diese Fehler im implementierten (kodierten) Zielprozess vorhanden sind und sich bei bestimmten Bedingungen im Versagen (f) bei der erwarteten Ausgabe des Prozesses manifestieren.

[0234] Die folgenden Definitionen sind an diesem Punkt notwendig:

Definition P1: Sei (s) ein Elementarprozess (Programmanweisung).

Definition P2: "Prozessadresse (PA)" {k,b} des Elements

– ist die Position dieses Elements innerhalb des "reduzierten Flussdiagramms" oder, was dasselbe ist, innerhalb des eX-Graphs dieses Prozesses.

[0235] (Bezugnahme auf **Fig.** 5(A) und 4) PA ist durch die zwei Koordinaten k und b definiert. Jede Anweisung (s) innerhalb des Prozesses hat ihre eigene PA, die durch {k,b} gemessen ist.

Definition P3: "Lokalzeit" {p}

– ist der Ganzzahlenwert des aktuellen Vorkommens (Vorhandensein) von Anweisung (s) innerhalb der Prozessausführung; (d. h. p = 1, 2, ...n)

[0236] Die eX-Maschine arbeitet nicht mit einer physischen Zeit. In dieser Erfindung berücksichtigt die Definition von Zeit, dass (s) Funktionalität den Prozess nur dann berührt, wenn und falls die Steuerung durch dieses (s) durchgeht, und es veranlasst, vorübergehend aktiv zu werden. Daher geben wir jedem Ereignis von (s) Ausführung seiner eigenen Zeitparameter.

[0237] Eine Lokalzeit wird ab dem Aufrufen des Prozesses gezählt.

Definition P4: "Prozessereignis" {k,b,p}

– ist ein Ereignis von (s)-Ausführung.

[0238] Es ist in der eX-Maschine durch drei Parameter {k,b,p} dargestellt, wo {k,b} den Platz ("Prozessadres-

se") identifiziert und {p} die Zeit "Lokalzeit" identifiziert.

Definition P5: "Prozesszeit"

– ist durch Prozessereignisse {k,b,p} repräsentiert.

[0239] Dies macht die Analyse unabhängig von Hardware- und betriebssystemabhängiger physischer Zeit, die für die Ausführung von spezifischen Prozesselementen notwendig ist.
[0240] Wir sagen, dass jeder Elementarprozess (s) innerhalb seiner eigenen Zeitkoordinate (p) lebt. Eine Zeit von einem Ansichtspunkt des Gesamtprozesses kann nur durch die Kombination von "Platz" und "Zeit" gemessen werden, d. h. "Prozessadresse" {k,b} und "Lokalzeit" {p}.
[0241] Falls wir beispielsweise auf die Prozesszeit zu dem Moment Bezug nehmen wollen, wenn die Anweisung A5 in **Fig.** 5(A) und **Fig.** 4 zum dritten Mal ausgeführt worden ist, würden wir auf diesen Moment als {12,2,3}-Ereignis Bezug nehmen.
[0242] Die Ausgabeereignisse des Prozesses sind die Gründe, warum der Prozess zunächst aufgebaut wird und üblicherweise nur falls diese Ausgabeereignisse inkorrekt sind, erklären wir, dass der Prozess Versagen zeigt.
[0243] Ein Computerprozess kann mehrere Ausgaben aufweisen, beispielsweise eine Ausgabe an das Systemende, andere an andere Dateien. Die eX-Analysemaschine wird automatisch unter Verwendung der inkrementellen systemausgabeprozesssynchronisierten (ISOPS) Map aktiviert, wie weiter erklärt werden wird.
[0244] Die Prozesszeit wird an der spezifischen Ausgabe nur durch die Abfolge an sie gerichteter Ausgabeereignisse dargestellt:
.. {ki,bi,pi} {kj,bj,pj} {kl,bl,pl} ..
[0245] Wenn man von der Ausgabe auf den Prozess blickt, sieht man solche durch die Ausgabeergebnisse ri rj r1 repräsentierten Prozessereignisse, wo die Prozesszeit von rj nach ri und vor r1 ist.
[0246] **Fig.** 12 stellt nur drei mögliche Arten von Versagen beim Erzeugen eines Ausgabeereignisses (r) zur Prozesszeit (t) dar.
[0247] Axiom 1: Für jede Ausgabe gibt es drei und nur drei Arten von Prozessversagen:
(f1) zur spezifischen Prozesszeit erwartetes Ausgabeereignis nicht erzeugt;
(f2) zur spezifischen Prozesszeit unerwartetes Ausgabeereignis wird erzeugt;
(f3) zur spezifischen Prozesszeit erwartetes Ausgabeereignis wurde mit falschem Wert erzeugt.

[0248] Die eX-Maschinenschnittstelle beinhaltet Befehle, welche diese drei Arten von Prozessversagen adressieren und daher kann die eX-Maschine jegliches Prozessversagen adressieren.
[0249] **Fig.** 13 zeichnet auf abstraktem Niveau das Prinzip auf, wie die eX-Maschine das Wissen des Prozesses über die Zeit akkumuliert. Diese Akkumulierung von Wissen wird im Detail unter Bezugnahme auf **Fig.** 7, **Fig.** 8, **Fig.** 14, **Fig.** 15, **Fig.** 16, **Fig.** 17 beschrieben.
[0250] Um **Fig.** 13 zu adressieren, sind die folgenden Definitionen notwendig:

Definition P6: "Korrekte Prozessadresse" (CPA)

[0251] Eine Prozessadresse PA wird als "korrekt" definiert, falls sie ein Prozesselement (s) enthält, das als ein "korrektes" Ereignis ausgeführt worden ist, d. h. ein korrektes Ergebnis am korrekten Ort und Zeit erzeugend)

Definition P7: "Unsichere Prozessadresse" (UPA)

[0252] Eine Prozessadresse PA ist "unsicher", wenn sie ein Prozesselement (s) enthält, das noch nicht in einem "korrekten" Ereignis ausgeführt worden ist.
[0253] Wenn wir die Prozessadresse als "korrekt" definieren, definieren wir ihre Inhalte als korrekt innerhalb des spezifischen Ereignisses. Beispielsweise mag eine Anweisung A = B + C nur am spezifischen Ort und spezifischer Zeit des Prozesses korrekt sein.
[0254] Es muss hier verstanden werden, dass die Verteilung von Lokalzeit von Prozessadressen während der Prozessausführung dynamisch vorgenommen wird und durch zwei Faktoren absolut festgelegt ist:
(1) Hartkodierte Prozesssteuerstruktur und Elementeinhalte,
(2) Ändern von Eingabewertenkombinationen;

[0255] So sind es unter der Bedingung unveränderter Prozessstruktur und der Inhalte der Prozesselemente die Eingabewerte, welche die Verteilung von Lokalzeit im Prozess definieren. Man beachte, dass wir nicht verlangen, dass das "korrekte" Prozesselement zu allen Zeiten korrekt ist.

[0256] Dies ist, wo die "Relativität der Korrektheit" ins Spiel kommt. Anders ausgedrückt, kann (s) korrekt sein nur relativ zu:

– Dem Ort und der Zeit (Prozesszeit),
– erwarteter Eingabe und
– erwarteter Ausgabe.

[0257] Diese Relativität ist genau der Grund für die Abwesenheit von Verfahren, die in der Lage sind, Programmkorrektheit zu beweisen. In unserer Sicht ist die einzige Antwort auf das Problem des Automatisierens des Prozesses des Lokalisierens der Fehlerposition die konstante Minimierung dieser Positionsunsicherheit.

Definition P8: "Ereigniskorrektheit"

[0258] Wir definieren ein Ausgabeereignis als "korrekt",. falls es ein "korrektes", d. h. erwartetes Ergebnis erzeugt. Diese Erwartung liegt in Prozesszeit und Wert.
[0259] Wir definieren ein Prozessereignis {k,b,p} außer dem Ausgabeereignis als "korrekt", falls es Teil der Kette von Ereignissen bildet, die zum Erzeugen des korrekten "Ausgabeereignisses" beitragen.
[0260] Das heißt, Ereignisse außer den Ausgabeereignissen werden durch "korrekte" Ausgabeereignisse als "korrekt" beansprucht.
[0261] Ereignisse können durch mehr als ein Ausgabeereignis als korrekt beansprucht werden. Beispielsweise sollen e1 e2 e3 e4 e5 e6 e7 eine Sequenz der Prozessereignisse repräsentieren, wobei e7 korrektes Ausgabeereignis ist, e1 und e3 Eingabeereignisse sind und e1, e2, e3, e4 und e6 beim Erzeugen des Wertes von e7 beteiligt waren. Dann sagen wird, dass die Ereignisse e1 bis e7, außer e5, korrekt sind.
[0262] Ereignis e5 kann von irgendeinem anderen korrekten Ausgabeereignis als korrekt definiert werden, in welchem die Berechnungskette e5 einen Platz einnehmen würde, aber bis dahin ist seine Korrektheit unsicher.
[0263] Diese Definition von Prozesselement-"Korrektheit" kann einfach durch die folgende Assoziation verstanden werden: Es sei angenommen, dass eine Computerplatine mit einem codierten Prozess assoziiert ist und jede integrierte Schaltung (IC) auf dieser Platine mit einem codierten Elementarprozess, wie einer Programmanweisung, assoziiert ist. Falls der IC nicht defekt ist und am korrekten Sockel platziert wird, annehmend, dass der IC teilhatte beim Erzeugen eins korrekten Ergebnisses bei der zuvor getesteten Eingangssignalkombination, wird dieser IC in der Zukunft beim Erzeugen desselben korrekten Ergebnisses bei derselben Eingabekombination teilhaben. Dieser IC ist daher als "korrekt" definiert, aufgrund seines POTENTIALS ein "korrektes" Ergebnis zu erzeugen, das am "korrekten" Ort auf der Computerplatine und bei der "korrekten" Eingabekombination positioniert ist, d. h. innerhalb von "korrektem" "Ort" und "Zeit".
[0264] Axiom 2: Die "Korrektheit" der Prozessfunktionalität ist unabhängig von ihrer "optimalen" Implementierung.
[0265] Axiom 3: Die Optimierung des Prozesses, der als korrekt definiert worden war, wird nicht seine Korrektheit, wie durch die eX-Maschine definiert, brechen.
[0266] Ein einfacher Beweis ist die Tatsache, dass von einer äquivalenten Optimierung nicht angenommen wird, dass sie die Prozessergebnisfunktionalität ändernt.

Definition P9: "Algorithmenraum"

– ist aus Prozessadressen {ki,bi} kombinierter Raum

[0267] Bezugnehme auf **Fig.** 5 und **Fig.** 4. Es ist wichtig, zu bemerken, dass bei den Bedingungen unveränderter Prozesskonstruktionen der Algorithmenraum konstant ist und durch die zwei Dimensionen des eX-Graphen des Zielprozesses – MK und NB – begrenzt ist.
[0268] In **Fig.** 5(A) und **Fig.** 4, welche den eX-Graph und das reduzierte Flussdiagramm unseres Probenprozesses repräsentieren, sind die Dimensionen (MK,NB) (14,5).

Definition P10: "Korrekter Algorithmenraum"

– ist aus PAs kombinierter Raum, die derzeit als CPAs definiert sind.

Definition P11: "Unsicherer Algorithmenraum"

– ist aus PAs kombinierter Raum, die derzeitig als UPAs definiert sind.

[0269] Wir verwenden den Begriff "derzeit definiert", weil die eX-Maschine eine Fähigkeit sowohl zum Annoncieren als auch zum Deannoncieren der Korrektheit von Ereignissen und PAs aufweist. Dieses partielle "Ver-

lieren von zuvor aufgebautem Wissen" ist unvermeidbar, wenn das Versagen in der Prozessausgabe wahrgenommen wird, die zuvor als korrekt akzeptiert worden ist.

Definition P12: "Ereignisdefinitionsraum"

[0270] eines Ereignisses (e) besteht aus PAs der (e)"Vorläufer"-Ereignisse.

Definition P13: "Vorläuferereignis", "Abkömmlingereignis"

[0271] Ereignis e1 ist ein "Vorläufer" von Ereignis e2, welches der "Abkömmling" von e1 ist, falls e1 beim Erzeugen von e2 beteiligt war entweder:

– teilgehabt hat am Erzeugen des Wertes von e2 (Wertvorläuferereignis); oder

– teilgehabt hat am Erzeugen eines Parameters für ein bedingtes Steuerungsereignis auf dem Weg nach e2 (Steuervorläuferereignis);

[0272] Bezugnahme auf **Fig.** 14: e(a) ist ein Nachfolger der Ereignisse, die in diesem Beispiel unter e(a) und innerhalb seiner Zweige gezeigt sind;
e(d) ist ein Abkömmling von e(i), e(j), e(k), e(l), e(m) und der Ereignisse, welche die Vorläufer von e(j) sind;
e(b) ist ein unmittelbarer Abkömmling von e(f) und e(g);
e(f) und e(g) sind unmittelbare Vorläufer von e(b);
[0273] Mit Bezug auf **Fig.** 13: Der Zweck dieses Diagramms ist lediglich, abstrakt zu zeigen, wie Prozessunsicherheit über die Zeit minimiert werden kann und wie die Position von Fehlern über die Zeit lokalisiert werden kann. Dieses Diagramm hilft beim Gesamtverständnis der eX-Maschine. Detailliertere Erläuterungen der eX-Maschine folgen.
[0274] Es gibt vier auf diesem Diagramm repräsentierte Objekte. Diese sind von links nach rechts:
Eingaberaum – alle möglichen Kombinationen von Eingabeparametern; d. h. falls der Prozess vier Eingabeparameter verwendet, ist ein Eingaberaum für diesen Prozess ein vierdimensionaler Raum;
unsicherer Algorithmenraum;
korrekter Algorithmenraum;
erwarteter Ausgaberaum – Kombination von erwarteten Ausgaberesultaten am untersuchten Ausgang;
t0, t1, t2, t3 repräsentieren verschiedene Prozesszeit, wobei t0 am Anfang des Prozesses ist, wenn noch keine Ausgabeergebnisse erzeugt sind und zu t1, t2 und t3 die Prozessausgabe kontinuierlich akkumuliert wird.
[0275] Sei es, dass das erste Versagen zur Prozesszeit t3 wahrgenommen wird. Dieses Diagramm zeigt, dass, wenn das eX-Maschinenverfahren der Unsicherheitsminimierung angewendet wird, der Bereich der mit der Position von Fehler(n), die für das Versagen zur Prozesszeit t3 verantwortlich sind, assoziierten Unsicherheit kleiner ist als, falls der Fehler zur Prozesszeit t2 erzeugt worden wäre usw.
[0276] Bei der Zielprozessausführung ist ein Eingaberaum mit echter Eingabe belegt und ein Ausgaberaum ist mit echter Ausgabe belegt.
[0277] Mit der Akkumulierung der Ausgabeergebnisse expandiert die eX-Maschine, die durch die weiter beschriebenen Verfahren ausgebaut wird, korrekten Algorithmusraum und vermindert ungewissen Algorithmusraum. Dies ist so, da die Summe der zwei gleich dem Algorithmenraum ist, der in dem Moment, zu dem der Zielprozess kodiert wird, definiert wird und der gleich bleibt, solange der Zielprozess unverändert bleibt.
[0278] Wir proklamieren, dass das Folgende während der Zielprozessvorwärtsausführung, die durch die eX-Maschine analysiert ist, gültig ist:

a: Unsicherer Algorithmenraum wird konstant mit Vermehren des korrekten Algorithmusraums vermindert.

b: Ereignisdefinitionsraum für jedes Prozessausgabeereignis ist beschränkt; die Grenze ist ein gesamter Algorithmusraum des Prozesses. Daher wird kontinuierlich weniger und weniger Anteil des Ereignisdefinitionsraums innerhalb des Bereichs des unsicheren Algorithmusraums während der Vorwärtsausführung des Zielprozesses liegen.

[0279] Unter Bezugnahme auf **Fig.** 13, falls d(e2) der Ereignisdefinitionsraum eines unerwarteten Ausgabeergebnisses (e2) zur Prozesszeit t2 ist und d(e3) der Ereignisdefinitionsraum eines anderen unerwarteten Ausgabeereignisses (e3), dann werden die Bereiche von Fehlerpositionsunsicherheit zu Prozesszeit t2 und t3 durch Teile von d(e2) und d(e3) repräsentiert, die durch "//" schattiert sind.
[0280] Der Bereich von Fehlerpositionsunsicherheit ist zur Prozesszeit t3 kleiner als der entsprechende Bereich zur Prozesszeit t2.
[0281] Obwohl ein Ereignisdefinitionsraum potentiell mit der Vorwärtsausführung des Zielprozesses wächst, gestattet die eX-Maschine eine konstante Minimierung von Fehlerpositionsunsicherheitsbereichen.
[0282] Die Ergebnisse der eX-Maschinenanalyse werden auf unsere Definition von korrekter Prozessadresse (CPA) bezogen, die sowohl zeitsensitiv als auch ausgabeergebniserwartungssensitiv ist.

[0283] eX-Maschinenprinzip des Lokalisierens von Fehlerpositionen oder einem Ort für notwendige Modifikationen:
[0284] Die für das Versagen eines Ausgabeergebnisses verantwortlichen Fehler oder ein Ort für Modifikationen aufgrund neuer Systemanforderungen, sind innerhalb desjenigen Teils des Ausgabeergebnisereignisdefinitionsraums lokalisiert, der innerhalb des Bereichs unsicheren Algorithmusraums liegt.
[0285] Wenn wir davon sprechen, dass die Fehlerposition auf mehrere UPAs beschränkt ist, meinen wir, dass die einzigen gestatteten Modifikationen innerhalb der erwähnten UPAs liegen. Ansonsten muss der neue Ereignisdefinitionsraum durch Expandieren des Algorithmenraums aufgebaut werden, d. h. durch Erzeugen neuer PAs. Daher bildet unsere Definition von Prozessunsicherheit die Basis für das Vorwärtskonstruieren des. Zielprozesses, d. h. die Basis für die automatisierte Vorwärtsprogrammentwicklung. Dies bedeutet, dass es mit der eX-Maschine möglich wird, automatisch den Bereich zu lokalisieren, zum Aufbauen auf dem existierenden Prozess in solch einer Weise, dass wir nicht rückgängig machen und zerstören, was während der Prozesswartung und Vorwärtsprogrammentwicklung "korrekt" arbeitet.
[0286] Man nehme z.B. an, dass es 50 Ereignisse erforderte, die an 12 Prozessadressen stattfanden, um Ausgabeereignis w(k,b,p) zu erzeugen. Mögen 45 dieser Ereignisse an 10 Prozessadressen stattgefunden haben, die als korrekt bekannt sind. Dann haben wir 5 Ereignisse, die an zwei unsicheren Prozessadressen stattfanden, die an der Erzeugung von w(k,b,p) beteiligt waren.
[0287] Falls w(k,b,p) korrekt (erwartet) ist, dann werden diese fünf Ereignisse durch die eX-Maschine als korrekt (bestätigt) gesetzt werden und zwei Anweisungen werden auf den Status von "korrekte Elementarprozesse" aktualisiert und ihre Prozessadressen werden zum Status von CPAs aktualisiert.
[0288] Falls w(k,b,p) ein inkorrektes (unerwartetes) Ergebnis war, wird der Bereich von Unsicherheit der Fehlerposition diese zwei PAs sein. Das heißt, dass das erwartete Ergebnis erzielt werden kann durch entweder: () Modifikation dieser zwei. Anweisungen oder () Aufbauen eines neuen Definitionsraums für dieses w(k,b,p);

4 Definition von eX-Maschinenprinzipien der Zielprozesswissenspropagierung (Wissensinduktion);

[0289] Zielprozess-"Wissensverbreitung" oder "Wissensinduktion" ist der Prozess, der zur Expansion des korrekten Algorithmusraums führt.
[0290] Die Beziehung zwischen A, L, D, E, X, *, +, C, !, /, R, =, S und W-Elementen des eX-Objektsatzes sind solcher Art, dass das "bestätigte" Ereignis der W-Elementausführung die Macht hat, die Kette von sekundären Prozesses zu aktivieren. Diese sekundären Prozesse werden die Ereignisse der Ausführung solcher eX-Graphelemente "genehmigen", die bei der Erzeugung des "bestätigten" W-Ereignisses beteiligt waren.

Definition K1: Ereignis "Bestätigung"

– ist ein Prozess des Validierens der Ereignisse "Korrektheit" durch die eX-Analysemaschine, wobei die Ereigniskorrektheit in P8 definiert ist.

Definition K2: "Steuerungsereignis"

– ist ein Ereignis, das definitiv ("unbedingtes Steuerereignis") oder potentiell ("bedingtes Steuerungsereignis") die Prozessrichtung vom Herunterfallen gemäß dem eX-Graphen und dem reduzierten Flussdiagramm ändern wird.

Definition K3: "Unbedingtes Steuerereignis"

– ist ein Ereignis der Ausführung von Rückspul- oder Ausgangsenden {*,+, C, R=}

Definition K4: "Bedingtes Steuerereignis"

– ist ein Ereignis der Ausführung eines L-Element oder Evaluierung einer Schleifenausgangsbedingung von D-Element.

[0291] Eine positive Lösung eines bedingten Steuerungsereignisses wird zur Steuerungsverschiebung nach rechts gemäß dem eX-Graphen und reduzierten Flussdiagramm führen. Eine negative Lösung eines bedingten Steuerungsereignisses wird dazu führen, der Steuerung zu gestatten, gemäß dem eX-Graph und dem reduzierten Flussdiagramm herunterzufallen.

Definition K5: "Datenmanipulationsereignis"

– ist ein Ereignis der Ausführung von A oder E oder dem arithmetischen Teil eines D-Element des eX-Objektsatzes;

Definition K6: "Anfangsdatenmanipulationsereignis"

– ist eines der Folgenden: E-Ereignis, A-Ereignis der Dateninitialisierung oder D-Ereignis der Schleifenindexinitialisierung;

Definition K7: "Sekundäres Datenmanipulationsereignis"

– ist ein anderes Datenmanipulationsereignis als das Anfangsdatenmanipulationsereignis.

[0292] Die eX-Maschine gestattet die Akkumulierung von Wissen über das Thema, welche Ereignisse die "korrekte" Prozessausführung bilden, durch Aufbauen der ACP-analysierten Kalkulationsbasis.
[0293] Das grundlegende Prinzip hinter ACB ist, dass das Wissen über die "korrekten" Ausführungsereignisse bewahrt werden muss.
[0294] Die "Korrektheit" wird als ein POTENTIAL zum Erzeugen "korrekter" Ereignisse in Ort, Zeit und Wert nach Bedingung von "korrekten" Eingabeereignissen verstanden. Somit gestattet die analysierte Kalkulationsbasis, das POTENTIAL der "korrekten" Ausführung zu bewahren.
[0295] Aufbauen der analysierten Kalkulationsbasis ist unsere Implementierung des Prinzips, von dem wir glauben, dass es für jeden Prozess universell ist: Wenn der Zielprozess "korrekte" Ausgabeereignisse anhand der validen Eingabedatenkombination erzeugt, muss das Wissen über solche Ereignisdefinitionsräume bewahrt werden. Die nachfolgenden Ausgabeereignisse können den Ereignisdefinitionsraum von bereits bestätigten Resultatereignissen (W-Ereignissen) verwenden, aber nicht ändern, und werden daher nicht die zuvor korrekte Zielprozessausführung ändern.

Definition K8: Ein elementares "Wissen" über den Elementarprozess (s)

– wird erhalten, wenn wir eines seiner Ausführungsereignisse "bestätigen".

[0296] Akkumulieren des "Wissens" über den Elementarprozess (s) bedeutet daher das Akkumulieren von Information über die (s)-"Korrektheit" oder seine Unsicherheit innerhalb seiner verschiedenen Ausführungsereignisse;
[0297] Ein elementares "Wissen" über Elementarprozess (s) besteht daher aus den folgenden Komponenten:
a – das Wissen über (s)-Inhaltekorrektheit;
b – das Wissen über Prozesszeit (Prozessort und Lokalzeit), wo ein Ereignis der (s)-Ausführung ein "korrektes" Ereignis bildete oder nicht bildete.

Definition K9: "Elementare Wissensverbreitung", oder "Elementare Wissensinduktion":

[0298] Die Tatsache, dass wir wissen: a) Die Inhalte, und
b) die Prozesszeit eines bestätigten Ereignisses:
gestattet uns, das "elementare Wissen" über dieses Ereignis an verschiedene Prozessadressen durch automatisches Lokalisieren und Analysieren von "Vorläufer"-Ereignissen den ganzen Weg zurück zu "Wissensverbreitungsendereignissen" zu verbreiten.

Definition K10: "Wissensverbreitungsende"-Ereignis (KP-Ende)

– ist eine der folgenden zwei Arten von Ereignissen:
1) Ereignis, das keinen Vorläufer innerhalb des Zielprozesses hat, d. h. ein Anfangsdatenmanipulationsereignis;
2) zuvor bestätigtes Ereignis;

[0299] Zuvor bestätigte Ereignisse stabilisieren den Prozess der Wissensverbreitung und hindern sie daran, immer komplexer mit der Akkumulierung von Vorläuferereignissen während der Zielprozessvorwärtsausführung zu werden. Sobald der "Wissensverbreitungs"-Prozess ein zuvor bestätigtes Ereignis trifft, können wir sicher sein, dass alle seine Vorläuferereignisse bereits zuvor bestätigt worden sind.
[0300] In der folgenden Beschreibung der Evaluierung der "Unsicherheit" innerhalb des Zielprozesses, d. h.

einem "Wissensdeduktions"-Prozess im Gegensatz zum hier beschriebenen "Wissensinduktions"-Prozess werden dieselben "Wissensverbreitungs"-Enden zu den "Unsicherheits"-Enden.
[0301] Bezugnahme auf **Fig.** 14: In diesem Beispiel gibt es zwei W-Ereignisse: e(a) und e(n), die zusammen mit einigen ihrer Vorläufer repräsentiert sind. W-Ereignisse sind in der Richtung der Prozessausführung bestätigt, in **Fig.** 14 als --> gezeigt. Nur ein Teil des Ereignisdefinitionsraums wird für jedes der zwei W-Ereignisse gezeigt.
[0302] Möge *e ein "KP-Ende"-Ereignis bezeichnen, d. h. ein Anfangsdatenmanipulationsereignis und #*e ein "KP-Ende"-Ereignis bezeichnen, d. h. ein zuvor bestätigtes Ereignis.
[0303] In diesem Beispiel wurde der Wert des W-Ereignisses e(a) aus drei Parametern berechnet, deren Werte wiederum durch die folgenden Ereignisse erzeugt wurden: e(b), e(c) und e(d).
[0304] Der Wert von e(b) wurde aus zwei Parametern berechnet, deren Werte wiederum durch die Ereignisse produziert wurden: e(f) und e(g);
[0305] Ereignis e(f) hat keine Vorläufer, da es ein Anfangsdatenmanipulationsereignis ist und daher ein KP-Ende.
[0306] Ereignis e(g) wurde zuvor bestätigt und ist daher auch ein KP-Ende.
[0307] Wir versuchen nicht, die Vorläufer von KP-Enden zu lokalisieren, da sie entweder nicht vorhanden sind, wie im Falle von e(f), oder zuvor bestätigt, wie im Fall von e(g). Dies ist, worauf das "beendete Induktions"-Prinzip basiert.
[0308] Da die Richtung der Prozessausführung vom W-Ereignis e(a) zum W-Ereignis e(n) verläuft, wird ein Ereignis e(i) ein KP-Ende im Prozess des Verbreitens von Elementarwissen vom W-Ereignis e(n), das später passiert, da e(i) und seine Vorläufer bereits während der Wissensverbreitung von e(a) bestätigt worden sind.

Die Regel der Wissensverbreitung:

[0309] Der Prozess des Bestätigens eines Ereignisses erzeugt einen sekundären Prozess des Bestätigens seiner unmittelbaren Vorläufer, solange nicht:

a) die unmittelbaren Vorläufer bereits bestätigt sind, oder
b) abwesend (in einem Fall, wenn das Ereignis ein Anfangsdatenmanipulationsereignis ist)

[0310] Daher wird die Ausbreitung von Elementarwissen, "Wissensinduktion" eine Kettenreaktion, deren Expansion durch die Wissensverbreitungsenden gesteuert wird.
[0311] Unter Bezug auf das Beispiel in **Fig.** 17: Ereignis (e0), das die Prozesszeit (k0,b0,p0) ist, ist das erste bestätigte Ereignis der (s0)-Ausführung, die den Wert von (y) berechnet, da die PA der Anweisung (s0) den Status von UPA aufweist. Daher wird PA von (s0) zu CPA aktualisiert.
[0312] Die unmittelbaren Vorläufer sind die Ereignisse der Berechnung der Parameter, die bei der Kalkulation des (y)-Wert beteiligt waren: x1, x2, x3. Man nehme an, dass die Werte von x1 und x2 unmittelbar vor dem Ereignis von (k0,b0,p0) zu Prozesszeiten (k1,b1,p1) und (k2,b2,p2) erzeugt wurden. Man nehme an, dass sie zuvor bestätigt wurden und daher behandeln wir sie als KP-Enden und prüfen ihre Vorläufer nicht.
[0313] Man nehme an, das Ereignis des Kalkulierens von x3 unmittelbar vor (k0,b0,p0) gleich (k3,b3,p3) liegt und nicht zuvor bestätigt worden ist. Der Zustand CPA bedeutet lediglich, dass einige andere Ereignis(e) dieser PA-Ausführung zuvor bestätigt war(en). Wir werden fortfahren, seine unmittelbaren Vorläufer zu prüfen, das sind die Ereignisses des Berechnens der x3-Parameter, unmittelbar vor der Prozesszeit (k3,b3,p3). Usw. ...
[0314] Die Wissensverbreitung, aktiviert durch bestätigte W-Ereignisse, wird in der entgegengesetzten Richtung zur Richtung der Zielprozessausführung vorgenommen.

Definition K11: "Unmittelbarer Besitzer" eines Ereignisses:

[0315] e2 besitzt e1, falls e2 e1 bestätigt hat.;
[0316] In **Fig.** 14 wird e(i) durch e(d) besessen und nicht durch e(o).

Definition K12: "W-Besitzer" eines Ereignisses:

[0317] Jedes bestätigte Ereignis (e) hat einen W-Besitzer, der ein W-Ereignisabkömmling von (e) ist, das die Kette der Elementarwissensverbreitung begonnen hat, die (e) bestätigt hat.
[0318] Jedes Ereignis kann lediglich ein W-Besitzerereignis haben und nur ein unmittelbares Besitzerereignis, bei dem das W-Besitzerereignis genauso ein unmittelbarer Besitzer des Ereignisses sein könnte.
[0319] In **Fig.** 14 ist der W-Besitzer von e(i) e(a) und nicht e(n). Das W-Ereignis e(a) ist der W-Besitzer für e(b), e(f), e(d), e(i), e(k), e(m), e(l);
[0320] **Fig.** 15 zeigt, dass, wenn ein Ereignis (e) bestätigt wird, sein PA-Zustand überprüft wird und falls der Zustand UPA (unsichere Prozessadresse) ist, wird es auf CPA (korrekte Prozessadresse) aktualisiert. Das

W-Besitzerereignis von (e), welches die Prozesszeit {kW,bW,pW} ist, wird ein Attribut dieser CPA, verantwortlich zum Aktualisieren dieses PA-Zustands.

[0321] **Fig.** 16 stellt dieselbe Prozessadresse (k,b) zu verschiedenen Prozesszeiten t1, t2, t3, t4, t5 dar. Die Prozesszeiten t1, t2, t3, t4 und t5 beziehen sich auf die Ausführung anderer als der (k,b)-Elemente.

[0322] Auf die PA(k,b) von der Zeit t1 blickend, finden wir PA(k,b) im Zustand von (k,b,0), d. h. keine Ereignisse der (k,b)-Ausführung sind bereits erzeugt worden.

[0323] Zur Prozesszeit t2 finden wir PA(k,b) im Zustand (k,b,2), d. h., dass (k,b) zwei Ereignisse seiner Ausführung erfahren hat. Keines von ihnen ist bereits bestätigt worden. PA(k,b) hat einen Status von UPA.

[0324] Zur Zeit t3 finden wir PA(k,b) im Zustand (k,b,3), d. h. es gab drei Ereignisse von (k,b)-Ausführung: (k,b,1), (k,b,2) und (k,b,3) und Ereignis (k,b,2) wurde bestätigt. Der Zustand von (k,b) wurde beim W-Besitzerattribut von UPA auf CPA aktualisiert, das der W-Besitzer von (k,b,2) war.

[0325] Während der weiteren Bestätigung anderer (k,b)-Ausführungsereignisse bleibt das W-Besitzerattribut unverändert und ist der W-Besitzer von (k,b,2).

[0326] Das W-Besitzerattribut von CPA gestattet es, das Wissen über den Zielprozess während der Variationen bei der Ausgabeergebnisakzeptanz aktuell zu halten.

[0327] Wenn eines der zuvor bestätigten W-Ereignisse aberkannt wird, fällt das Zielprozesswissen auf das Niveau, das durch die Position des aberkannten W-Besitzerereignisses in der Zielprozesszeit bestimmt ist. Wir nennen diesen Prozess der eX-Maschine: "Verlieren des Wissens".

5 Aufbau des eX-Modells des Zielprozesses;
Definition der analysierten Kalkulationsbasis (ACB);
Aufbauen des eX-Verwahrungsorts des Zielprozesses;

Aufbauen des eX-Modells des Zielprozesses:

[0328] Bezugnahme auf **Fig.** 7: Das eX-Modell wird aus dem eX-Graphen als eine dreidimensional basierte Konstruktion aufgebaut, d. h. weitere Dimensionen könnten auf Basis der erwähnten drei hinzugefügt werden, wobei die dritte Dimension auf Basis der zwei Dimensionen des eX-Graphen aufgebaut wird.

[0329] Die dritte Dimension repräsentiert eine Struktur, die Attribute über Prozessadressen enthält. Verschiedene Arten von Attributen werden an verschiedene Prozessadressen (PA) gegeben, d. h. an verschiedene {k,b}-Koordinaten, abhängig von der Art des Elementarprozesses {A,L,D,W,E,*,/ usw.}, der unter dieser Prozessadresse gespeichert wird.

[0330] Jedes Element des eX-Modells, das der ausführbaren Anweisung entspricht, enthält drei Attribute, welche die Referenzen repräsentieren auf:

a) Eintragsnummer der entsprechenden Anweisung innerhalb des Zielprozesses;
b) Eintragsnummer der entsprechenden Anweisung innerhalb des eX-Verwahrungsortes; und
c) Eintragsnummer der entsprechenden Anweisung innerhalb des X-Quellcodes;

[0331] Dies etabliert die Referenz zwischen dem Zielprozess, seinem eX-Modell, dem eX-Verwahrungsort und dem X-Quellcode.

[0332] Für jedes ausführbare Prozesselement behält das eX-Modell ein Attribut der Lokalzeit;
Binäradressen (BA)-Attribut wird in dem Modell für jedes Element im eX-Graph dargestellt;
Rückkehrbinäradresse (RBA)-Attribut wird für alle Prozessenden dargestellt.

[0333] Den Rückspulenden (*), (C) oder (+) entsprechende PAs besitzen ein Attribut ihrer Rückspul-ID (ID/).

[0334] ID wird als eine Alternative zur {k,b}-Darstellung einer Prozessadresse verwendet. Jedes Paar {k,b} kann durch eine Ganzzahlwert-ID repräsentiert sein, welche die Anordnung dieser PA im eX-Graphenraum ist, wenn dieser Raum von (k = 1, b = 1) bis (k = MK,b = NB) durchlaufen wird und der k-Parameter zuerst inkrementiert wird.

ID = (b – 1)*MK + k, wobei MK das Maximal-k des eX-Graphen ist.

[0335] Somit weist in **Fig.** 5(A) das Graphelement *1, d. h. PA(9,1) eine ID = 9 auf und weist das Graphelement A4, d. h. PA(9,2), eine ID = 23 auf, da der MK = 14.

[0336] Das Folgende ist ein Beispiel der Implementierung der die dritte Dimension des eX-Modells für Prozessende bildenden Struktur, die 1 von {*|C| + |R| =} ist, gefolgt vom Beispiel der Implementierung der den ausführbaren Elementen eines Zielprozesses entsprechenden Struktur, dies ist eines von {A|L|D|E|W|X} und Änder-Steuerungsanweisung {S}.

[0337] Die Größe der Strukturen und die Größer ihres Elements sind implementationsabhängig und nur die für die eX-Maschinenimplementierung notwendigen Strukturelemente werden in diesen Beispielen gezeigt.

| * | ind | ID / | loc.time | BA | targ.r.# | Repos.r.# | X-Source r.# | RBA |

{C}
{+}

| R | | | loc.time | BA | targ.r.# | | X-Source r.# | |

{=}

[0338] Ausgangsenden {R,=} haben nicht die folgenden Attribute von Rückspulenden: {ind, ID/,Repos.r.#} ind – ist der Index des entsprechenden Prozesselements im eX-Rahmen und eX-Graphen;
[0339] Beispiele eines eX-Modellelements entsprechend PA(14,3) von **Fig.** 5(A):

| * | 2 | 6 | loc.time | 11 | targ.r.# | Repos.r.# | X-Source r.# | 2 |

repräsentiert binär 1011

repräsentiert binär 10

[0340] Das Lokalzeitattribut wird zum Zeitpunkt der Zielprozessausführung ausgefüllt.
[0341] Die Prozesselemente A,L,D,E enthaltenden PAs weisen "ACB-Zeiger"-Attribut auf, wobei ACB für analysierte Kalkulationsbasis steht.

| A | ind | ACB-ptr | loc.time | BA | targ.r.# | Repos.r.# | X-Source r.# |

{L}
{D}
{E}
{W}
{X}
{S}

[0342] Zusätzlich weisen die L- und D-Elemente ein den das Inkrement in der b-Koordinate durch die positive Lösung (Hochpotentiallösung) von L oder D repräsentierenden Wert definierte Attribut auf:

| ... | shift potential | ... |

[0343] Das ACB-ptr-Attribut von PA ist ein Zeiger auf die entsprechende ACB-Konstruktion, welche Informationen über die Geschichte der "bestätigten" Ausführungsereignisse des derzeitigen Prozesselements enthält.
[0344] Definition der analysierten Kalkulationsbasis (ACB):
die Sammlung der internen oder externen Speicherkonstruktionen, die jede einem anderen Element des Ziel-

prozesses entsprechen und Informationen enthalten, die den derzeitig bekannten Status dieses Elementverhaltens als "bestätigt" oder "unsicher" mit der Zeit bilden, wird in dieser Erfindung analysierte Kalkulationsbasis (ACB) des Zielprozesses genannt.

[0345] Jede ACB-Konstruktion entspricht einer ausführbaren PA des Zielprozesses und kann als eine interne Variablenkonstruktion oder als eine Datei implementiert sein.

[0346] Die einzige Regel ist, dass jegliches Element oder Mitglied der ACB-Konstruktion ein Ereignis der Ausführung des Zielprozesselements entsprechend der Konstruktion repräsentiert. Die Lokalzeit des Ereignisses wird durch die Position innerhalb der Konstruktion repräsentiert.

[0347] Der Wert des ACB-Konstruktionselements kann auf einen von zwei Zuständen eingestellt werden: "Ein" und "Aus", wobei "Ein" dem bestätigten Ereignis entspricht und "Aus" dem Ereignis entspricht, das noch nicht bestätigt ist (d. h. unsicher).

[0348] Bezugnahme auf **Fig.** 19:

In unserer Implementierung weist ACB-ptr auf eine Binärdatei, die in zwei Teile unterteilt ist. Der erste ist der Kontrollbereich, der zwei Arten von Informationen hält: Nächstes-Parameter-Wort (NPW) und Unsicherheitsersatz-Wort (UDW). Der zweite Teil enthält einen Informationsbereich, der die "Wissenszustands"-Bits hält. Jedes Ausführungsereignis des Elements, das diesen ACB-ptr erzeugt hat, wird durch einen Bit im Informationsbereich repräsentiert, das ein "Wissenszustands"-Bit ist. Die Position des Bits relativ zum Anfang des Informationsbereichs der Datei entspricht der Lokalzeit des Ereignisses, und das Bit wird auf "Ein" oder "Aus" entsprechend dem jetzigen Zustand des "Wissens" über dem Ereignis "Korrektheits"-Zustand gesetzt.

[0349] Unterschiedliche Techniken der Dateikomprimierung könnten verwendet werden. In unserer Implementation hält UDW den relativen Versatz des ersten Bits, das im Zustand 0 ist. Bevor irgendein Ereignis der Elementausführung bestätigt wird, ist das entsprechende UDW gleich 0, das erste Bit des Informationsbereichs entspricht dem ersten Ereignis der Elementausführung und keine Informationsbereichskomprimierung hat stattgefunden.

[0350] Wenn zu einem späteren Zeitpunkt UDW gesetzt wird, beispielsweise auf 2000, würde es repräsentieren, dass die erste Bitposition des Informationsbereichs der Lokalzeit **2001** entspricht und alle 2000 vorherigen Ereignisse dieser Elementausführung bestätigt wären. Daher wird die dem erwähnten ACB-Zeiger entsprechende Binärdatei durch Verschieben um 2000 Bit-Positionen nach links komprimiert, da wir wissen, dass die ersten 2000 Positionen mit 1 gefüllt würden.

[0351] Der Wert von UDW kann sich in beiden Richtungen während der "Induktion von Wissen" oder "Verlieren von Wissen" Prozessen der eX-Maschine ändern, die das derzeitige Zielprozesselement analysiert. Dies würde durch Verschieben von Bits innerhalb des Informationsbereichs der entsprechenden ACB-Konstruktion erreicht werden.

[0352] Beispielsweise wird zu der Zeit, wo die ersten und vierten Positionen des D1 ACB Informationsbereich ihren Zustand während des "Wissensinduktions"-Prozesses auf "Ein" ändern würden, die UDW von D1 auf die dezimale 7 gesetzt und ihre Informationsbereichbits würden um 7 Positionen nach links verschoben werden.

[0353] Jeder Parameter einer Eingabenanweisung {E} und jeder Parameter einer Aufrufanweisung {X} wird durch ihre ACB-Konstruktion repräsentiert.

[0354] ACB-ptr des entsprechenden Modellelementes zeigt auf die dem ersten Parameter entsprechende ACB-Konstruktion. Das Nächste-Parameter-Wort (NPW) dieser Konstruktion zeigt auf die ACB-Konstruktion des nächsten Parameters, falls er anwesend ist, usw. Ansonsten ist NPW leer.

[0355] Die ACB eines Zielprozesses, die fünf ausführbare Elemente {A1,A2,D1,E1,X1} enthält, wo E1 zwei Eingabeparameter aufweist und X1 drei Aufrufparameter aufweist, kann wie die durch **Fig.** 19 repräsentierte aussehen.

AUFBAUEN EINES eX-AUFBEWAHRUNGSORTES DES ZIELPROZESSES:

[0356] Bezugnahme auf **Fig.** 7: Ein eX-Verwahrungsort des Zielprozesses wird zur selben Zeit wie der eX-Graph (oder eX-Rahmen, falls dieser Schritt implementiert war) aufgebaut. Jedes Element des einer ausführbaren Anweisung (A,L,D,E,W,X,S-Elemente des eX-Objektsatzes) entsprechenden Hauptzweigs wird in der eX-Verwahrungsortdatei gespeichert.

[0357] Das Volumen des in dem eX-Verwahrungsort enthaltenen Codes ist kleiner als der entsprechende Zielprozessquellcode. Dies ist so, weil der eX-Verwahrungsort nicht irgendwelche Prozesssteuerungsanweisungen enthält, wobei wir unter Prozesssteuerungsanweisungen Prozessanweisungen verstehen, die nicht die Datenmanipulation oder Validierung involvieren, d. h. der eX-Verwahrungsort enthält keine unbedingten Verzweigungen, Stop, Rücksprung, then, else, endif usw.

[0358] Zu der Zeit, wenn der Algorithmus aus dem Zielprozess durch das zuvor beschriebene "nächst niedrige Potentialprinzip" extrahiert wird, wird die Steuerungsstruktur in der Struktur des eX-Graphen und eX-Modell bewahrt, aufgebaut auf seiner Basis (**Fig.** 7).

[0359] Daher wird der eX-Verwahrungsort alle Prozesselemente enthalten, aber nicht die Definition der Pro-

zesssteuerung, die durch das eX-Modell verwahrt wird.

Beispiel 1: für die Zielprozessanweisung: if(d<10) go to label

wird der Verwahrungsort enthalten: if(d<10)

Beispiel 2: für die Zielprozessanweisung:

```
if(a.gt.b)then
        c=d
    else
        g=f(g)
endif
return
```

wird der Verwahrungsort enthalten:

```
if(a.gt.b)
c=d
e=f(g)
```

[0360] Jede Anweisung des eX-Verwahrungsorts wird im Attribut der entsprechenden Prozessadresse {k,b} innerhalb des eX-Modells referenziert. Beispielsweise muss, falls der Verwahrungsort als RRDS (relativer Aufzeichnungsdatensatz) gehalten wird, das entsprechende Referenzattribut die relative Eintragsnummer der entsprechenden Anweisung innerhalb des Verwahrungsortes enthalten. Man beziehe sich auf die im Abschnitt "Aufbauen des eX-Modells des Zielprozesses" erwähnte Verwahrungsorteintragsnummer (Repos.r.#).

6 Aufbauen von X-Quellcode;
Minimalüberbauinstrumentation für das Erfassen der Zielprozessausführungsgeschichte.
Aufbauen einer Anwender-eX-Maschinen-Zielprozessschnittstelle mittels der inkrementellen Systemausgabeprozesssynchronisierten (ISOPS)-Karte;
Aufbauen der Systemeingabeprozess-synchronisierten (SIPS)-Karte;

Aufbauen des X-Quellcodes

[0361] Um den Zielprozess, der von dem traditionellen Objektcode ausgeführt wird, durch die eX-Maschine zu analysieren, bauen wir den X-Quellcode aus dem eX-Modell und aus dem eX-Verwahrungsort jedes Zielprozesses (Hauptprogramm, Unterroutine oder Funktion) des Zielsystems von Programmen.
[0362] Das Ziel des Erschaffens des X-Quellcodes liegt im Instrumentieren der Synchronisierung zwischen dem Zielprozess und dem eX-Maschinenprozess des Akkumulierens des Wissens des Zielprozessverhaltens. Neben instrumentierten Synchronisationssignalen wird sich der X-Quellcode möglicherweise vom ursprünglichen Zielprozessquellcode darin unterscheiden, dass der X-Quellcode keinen "toten Code", d. h. unerreichbaren Code, enthält, und darin dass der X-Quellcode potentiell gemäß dem ursprünglichen Zielquellcode restrukturiert sein kann, falls der zweite nicht gemäß den eX-Modellregeln strukturiert war, beispielsweise falls er zusätzliche GO TOs hatte.

Bezugnahme auf **Fig.** 7 und 8:

[0363] X-Quellcode, der von X-Quellgenerator (**17** in **Fig.** 7) aus zwei Objekten erzeugt wird: eX-Modell (**8** in **Fig.** 7) und eX-Verwahrungsort (**10** in **Fig.** 7) werden wiederum das eX-Modell durch Synchronisationssignale (**4**) "fahren", die zu der eX-Schnittstelle geleitet werden, die wiederum durch Signale (**6**) die Analysemaschine aktiviert, welche das eX-Modell durch Befehle (**7**) "fährt".
[0364] Da der X-Quellcode aus dem eX-Modell und dem eX-Verwahrungsort aufgebaut ist, die wiederum aus der Quelle des Zielprozesses aufgebaut sind, gibt es 1 : 1-Funktionsäquivalente zwischen dem Zielprozess, dem eX-Modell und dem X-Quellcode.
[0365] Um X-Quellcode zu erzeugen, wird das Folgende gemacht:
– die Inhalte jeder ausführbaren Ausweisung für den Quellcode werden aus dem eX-Verwahrungsort (**10** in **Fig.** 7) genommen;

– die Steueranweisungen werden aus dem eX-Modell (**8** in **Fig.** 7) konstruiert;
– die Ziel-Modell-Synchronisationssignale werden instrumentiert; (man verweise beispielsweise auf **Fig.** 20 und 21).

[0366] Wir werden das Aufbauen der Synchronisationsschnittstelle zwischen dem Zielprozess und der eX-Analysemaschine am Beispiel der "verzögert sequentiellen" Modellsynchronisation erläutern, die im folgenden Punkt definiert ist.
[0367] Die Synchronisationssignale werden hierdurch CALL-Anweisungen (**4** in **Fig.** 8) erzeugt, die an die eX-Schnittstelle (eXI) (11) drei Parameter übergeben: (k,b,m), wobei {k,b} ein Paar ist, das die Prozessadresse definiert, und (m) ein Modell des Zielprozesses definiert, d. h. definiert, welcher Zielprozess derzeit ausgeführt wird.
[0368] Anstelle von drei Parametern (k,b,m) könnte die Schnittstelle durch zwei Parameter implementiert sein: (IDT,m), wobei IDT eine alternative Darstellung der Prozessadresse des Elements ist und aus den (k,b)-Coordinaten des entsprechenden Endes durch die zuvor beschriebene Formel berechnet wird:

IDT = (b – 1)*MK + k; ("Formel der ID"),

wobei MK die maximale k-Koordinate des Zielprozessmodells ist.

Minimale Überbauinstrumentierung zum Erfassen der Zielprozessausführungsgeschichte.

[0369] Der Endensynchronisierungsaufruf wird vor jeder ein Prozessende (Definition F3) repräsentierende Anweisung hinzugefügt.
[0370] Beispiel von **Fig.** 4: Unter der Annahme, dass wir eine X-Quelle für den Prozess aufbauen, die innerhalb des Systems von Programmen durch das Modell identifiziert ist, m = 2, wenn das GO TO Ende (a) vom Ende von Zweig b = 1 repräsentiert wird:
call eXI(9,1,2)
go to 'LABEL a'
und vom Ende des Zweigs b = 5:
call eXI(10,5,2)
go to 'LABEL a'
[0371] Die Syntax von 'LABEL a' unterscheidet sich in unterschiedlichen Programmiersprachen. Dieses Verfahren des Aufzeichnens der Prozessausführungsgeschichte repräsentiert den minimalen Überbau (minimale Instrumentierung), wie als Nächstes erläutert wird.
[0372] Wenn eXI einen Aufruf empfängt, wobei die PA durch eXI als ein Ende identifiziert wird, nimmt es diese PA gemeinsam mit {m} auf, wobei {m} der zuvor beschriebene Modellidentifizierer in der Systemausführungsgeschichtsdatei (**3** in **Fig.** 8) ist.
[0373] In unserer Implementierung der Synchronisation des Endes werden die Koordinaten des Endes {k,b} durch ihre IDTs repräsentiert.
[0374] Bezugnahme auf **Fig.** 8: eX INTERFACE (eXI)-Modul übersetzt IDTs in Paaren (k,b) durch die zuvor beschriebene "Formel der ID".
[0375] Durch Lokalisieren der Koordinaten (k,b) im Modell identifiziert das eXI-Modul, welches Ende das Synchronisierungssignal geschickt hat.
[0376] Die Sequenz der ausgeführten Prozessenden ist die hinreichende und notwendige (minimaler Überbau) Information, um die Zielprozessverhaltensgeschichte für irgendein gegebenes Modell eines Zielprozesses wieder zu erschaffen.
[0377] Erläuterung: Innerhalb unseres Modells gibt es nur einen Durchgang zwischen dem START und spezifischen Ende oder zwischen einem Eintrag einer Rücksprung-ID des vorherigen Endes und dem nächsten Ende. Die Rückspul-ID (ID/) ist als ein Attribut des Endes des eX-Modells gespeichert worden, wenn das Modell gebaut wird.
[0378] Wenn eXI eine Information über das nächste ausgeführte Ende empfängt, befiehlt es der Analysemaschine, den Teil des derzeitig ausgeführten Durchgangs innerhalb des Zielprozesses zurück zu verfolgen, das heisst von der Rückspul-ID des zuvor ausgeführten Endes (oder vom START, falls das derzeitig ausgeführte Ende das erste war, seit der Prozess betreten wurde) zum derzeitigen Ende. Jeder Durchgang durch die PA repräsentiert ihr Ausführungsereignis und inkrementiert ihr Lokalzeitattribut (p) um 1.
[0379] Man nehme beispielsweise an, dass die folgenden Signale von eXI während der Ausführung des aus dem eX-Graphen in **Fig.** 4 aufgebauten X-Quellcodes empfangen worden sind: 45,9,55.
[0380] Dies sind die IDTs der folgenden PAs: {14,3}, {9,1}, {10,4}, die uns sagen, dass der Zielprozess auf der folgenden Route ausgeführt wurde:

A1 A2 L1(-)A3 L2(+) A4 X1 D1(+) A6 W1 A3 L2(-) A2 L1(+) W2 E1 L3(-

) A7 STOP

[0381] Um die Repetition desselben Endes zu repräsentieren, wie es oft in DO, FOR, WHILE usw., Schleifen der Fall ist, verwenden wir eine negative Zahl. Beispielsweise würde 27–5 42 bedeuten, das die DO Schleife D1 6-mal eine negative Entscheidung getroffen hat (durch ihren Körper gegangen ist) (hat das Ende mit IDT = 27 erreicht), bevor sie den DO-Zyklus verlassen hat.
[0382] Im vorherigen Beispiel hatten wir drei Synchronisierungssignale: 42,9,52, um 29 Ereignisse der Prozessverhaltensgeschichte zu repräsentieren. Andere Optionen des Repräsentierens der Prozessgeschichte können entweder sein:

– Aussenden des Synchronisationssignals nach jedem ausgeführten Element, wie es durch einige Debug-Pakete praktiziert wird, welches in unserem Beispiel 29 Signale erzeugt haben würde; oder
– Aufzeichnen der aus durchlaufenden LCs bestehenden Paare gemeinsam mit ihren Lösungen: {L1,0} {L2,1} {D1,1} {L2,0} {L1,1} {L3,0}, d. h. 12 Signale.

[0383] Wir haben gerade demonstriert, dass das Aufzeichnen der Prozessausführungsgeschichte durch Aufzeichnen der Sequenz der ausgeführten Prozessenden die notwendige und hinreichende Information über die Zielprozessdynamik ist. Sie implementiert den Minimalüberbau sowohl in der Zahl der Synchronisierungssignale als auch der Zahl der Instrumentierungseinsätze, die nötig sind, um die Zielprozessdynamik unter Verwendung traditionellen Objektcodeprozessierens zu erfassen. Wie wir weiter zeigen werden, ist diese Instrumentierung nicht notwendig, wenn man eX-Objektcode verwendet.
[0384] Aufbauen der Anwender-eX-Maschinen-Zielprozessschnittstelle mittels einer inkrementierten Systemausgabeprozess-synchronisierten (ISOPS)-Karte.
[0385] Das Folgende beschreibt eine Schnittstelle zwischen einem Anwender und einem Zielprozess durch die eX-Maschine. Diese Schnittstelle ist, wie die Erfindung glaubt, das höchste mögliche Niveau an Schnittstelle, um eine Zielsoftware zu analysieren. Die eX-Maschinenanalyse wird durch eine Referenz auf die Systemausgabe aktiviert.
[0386] Es sind die Systemausgabeergebnisse, die der Grund sind, aus dem wir Programme schreiben. Es sind die Systemausgabeergebnisse, die der Grund sind, warum wir Programme modifizieren oder reparieren möchten. Es sind die Systemausgabeergebnisse, existierend oder nicht-existierend, auf die wir uns in unserem Geist beziehen, wenn wir versuchen, einen existierenden Prozess zu verstehen oder versuchen, einen neuen Prozess aufzubauen. Akzeptanz von, Nichtakzeptanz von oder Referenz auf die Systemergebnisse ist das, was den tatsächlichen "Grund" für Prozessanalyse oder Prozesskonstruktion darstellt.
[0387] Hier vorgeschlagene Techniken gestatten es, der eX-Analysemaschine diesen "Grund" als einen Weg mitzuteilen, die Analyse zu aktivieren, wodurch weitere Zwischenschritte eliminiert werden. Das weiter beschriebene Verfahren gestattet es, eine Schnittstelle zwischen einem Anwender und der eX-Analysemaschine aufzubauen, die das Systemergebnis als einen Referenzpunkt verwendet, wie es intuitiv gemacht wird, um unsere kognitive Analyse durch ein menschliches Wesen zu aktivieren.
[0388] Die Inkrementierte Systemausgabeprozess-synchronisierte Karte (ISOPS-Karte) ist ein Referenzinstrument zwischen Ausgabesignalen und Prozessereignissen, die solche Signale erzeugt haben.
[0389] Die Folgende ist ein Beispielformat für die ISOPS-Karte. Es beseht aus 5 Arten von Einträgen {1.2.3.4.5}

Eintrag 1. definiert das W-Ereignis durch {k,b,p,m} Prozessparameter;
Eintrag 2. definiert Position {sl,sc} (Bildschirmzeile, Bildschirmspalte} des
Beginns des derzeitigen Ausgabesignals auf dem sysout-Bildschirm;
Eintrag. definiert ein Rechteck auf dem Systemausgabebildschirm {I1,c1,I2,c2}, welches das Rechteck mit der kleinsten Größe ist, das den Bereich den von der W-Ereignisausführung veränderten Bildschirms enthält;
Eintrag 4. enthält eine Ganzzahlinformation, die ein Versatz des entsprechenden Eintrags 5 vom Anfang der die veränderte Bildinformation (aus Einträgen 5 kombinierte Datei) enthaltenden Datei ist.
Eintrag 5. enthält den tatsächlichen Teil des sysout-Bildschirmpuffers, der dem Bereich von Eintrag 3 entspricht;

[0390] Da Eintrag 5 das Bild des veränderten Bereichs des Bildschirmpuffers und der einzige Eintrag variabler Größe ist, halten wir ihn in einer getrennten Datei – Incremental Image Monitor (IIM Datei in **Fig.** 8). Die anderen 4 ISOPS-Kartenparameter, die dem Eintrag 1, Eintrag 2, Eintrag 3 und Eintrag 4 entsprechen, werden in einer ISOPS-Kartenparameterdatei (ISPOS Par. in **Fig.** 8) gehalten. Die Kombination der ISOPS Parameterdatei mit der IIM Datei repräsentiert die Systemergebnisgeschichte (**2** in **Fig.** 8) und (**2** in **Fig.** 7).

ISOPS Kartenbeispielformat:

1. W-Ereignis	2. Bildschirm-position	3. geänderter Bildschirm-bereich	4. Versatz Bildmonitor-datei	in	5. inkrementelle Bildmonitor-datei
_____ISOPS Parameterdatei_____/					_IIM Datei_/
{k,b,p,m}	{sl,sc}	{l1,c1,l2,c2}	DDDDDD		'xxxxxxxxxxxxxxxx'
{k,b,p,m}	{sl,sc}	{l1,c1,l2,c2}	DDDDDD		'xxxxxxx'
{k,b,p,m}	{sl,sc}	{l1,c1,l2,c2}	DDDDDD		'xxxxxxxxxxxxxx'
{k,b,p,m}	{sl,sc}	{l1,c1,l2,c2}	DDDDDD		'xxxxxxx'
{k,b,p,m}	{sl,sc}	{l1,c1,l2,c2}	DDDDDD		'xxxxxxxxxxx'

[0391] Das Folgende beschreibt die zum Aufbauen der ISOPS Karte verwendeten Werte:

Eintrag 1., der das W-Ereignis {k,b,p,m} beschreibt, wird wie folgt erhalten: {k,b,m} sind die Synchronisierungsparameter (**4** in **Fig.** 8), die vom Zielprozess an eXI (**11** in **Fig.** 8) gesendet werden; eXI erhält {p} von Modell (**8**), entsprechend {m} als das derzeitige Zeitattribut der Prozesselementstruktur, die dieser {k,b}-Prozessadresse entspricht. Das Zeitattribut wird durch die Analysemaschine (**9**) aktuell gehalten, wie es im nachfolgenden Absatz beschrieben wird.

Eintrag 2. Bildschirm (Zeile und Spalte) {sl,sc} Position am Anfang jedes Systemausgabesignals wird wie folgt erhalten: Der {sl,sc} Eintrag ist gleich {1,1} für den allerersten W-Ereigniseintrag. Wenn eXI ein vom W-Ereignis gesendetes Synchronisationssignal empfängt, erhält eXI die derzeitige Bildschirmposition, d. h. im Effekt nach dieser W-Ereignisausführung. Diese Bildschirmposition wird weiter als {sl,sc} Eintrag für das nächste W-Ereignis verwendet.

Eintrag 3. ist das kleinste Viereck, das auf dem Bildschirm gezeichnet werden konnte, das den gegenüber der Prozesszeit des vorherigen W-Ereignisses veränderten Bildschirmbereich enthalten könnte. Dieses Rechteck ist durch den inkrementellen Bildextraktor/Konstruktor (**12** in **Fig.** 8) definiert, indem das vorherige Bildschirmbild (**15** in **Fig.** 8) das nach dem vorherigen W-Ereignis zur Prozesszeit gespeichert wird, mit dem derzeitigen Systemausgabebildschirmpuffer (**5** in **Fig.** 8) verglichen wird.

Eintrag 4. ist die relative Byte-Position des aktuellen Eintrags des inkrementellen Bildmonitors in der IIM-Datei, welche Position dem inkrementellen Bildextraktor/Konstruktor (IIE/C) (**12** in **Fig.** 8) bekannt ist. IIE/C kennt die derzeitige Position, da es die Länge in Bytes jedes Eintrags in der IIM-Datei kennt.

Eintrag 5. ist der derzeitige Eintrag in der IIM-Datei, der Teil des sysout Puffers ist, entsprechend dem Rechteck von Eintrag **3**.

Aufbauen der Systemeingabeprozess-synchronisierten (SIPS)-Karte;

[0392] Die SIPS-Karte wird aufgebaut, um in der Lage zu sein, für Analyse oder Abspielen auf eine spezifische Eingabe in das System zugreifen zu können, die zur spezifizierten Prozesszeit gemacht worden ist. Die SIPS-Karte wird in der SYSIN-Geschichtsdatei (**1** in **Fig.** 8) und (**1** in **Fig.** 7) gehalten. Für jedes Systemeingabeereignis wird ein Eintrag in der SYSIN-Geschichtsdatei vorgenommen. Der Eintrag besteht aus zwei Aufzeichnungen. Eine ist die tatsächliche Eingabezeichenkette und die andere ist ein Satz von Parametern {k,b,p,m}, welches die Prozesszeit {k,b,p} und das Modell {m}, das der derzeitigen Eingabekette entspricht, beschreibt.

[0393] Das eX-Schnittstellenmodul (**11** in **Fig.** 8) erzeugt einen Eintrag in der SYSIN-Geschichtsdatei aus den Synchronisationsparametern {ide,m} (**4** in **Fig.** 8) und einen Systemeingabepuffer (m) (**19** in **Fig.** 8). {k,b} wird aus {ide} durch die Formel der ID berechnet. Der Parameter {p} wird von der eXI aus der derzeitigen Prozesszeit innerhalb des eX-Modells erhalten.

[0394] **Fig.** 21 hat ein Beispiel des {ide,m} Synchronisationsausrufs CALL eXI (21,2), das vor der Eingabe von dem Ende (E1 in **Fig.** 20) instrumentiert wird. Sobald die Systemeingabe dem Zielprogramm bekannt ist, wird sie durch die instrumentierte nächste Anweisung "WRITE(M,002)CH" in der Pufferdatei (m) (**19** in **7**) aufgezeichnet und später in der SYSIN-Geschichtsdatei durch das eXI-Modul wieder aufgezeichnet. Die instrumentierte Anweisung, welche die Systemeingabe aufzeichnet, wird durch Konvertieren der Ein/Ausgabe-Aktion

von "READ" nach "WRITE" gebaut, die an die Monitordatei {m} gerichtet ist.

7 Beispiel für Zielprozess, sein eX-Modell, eX-Verwahrungsort und X-Quellcode;

[0395] Bezugnahme auf **Fig.** 20, **Fig.** 21 und **Fig.** 7:
[0396] **Fig.** 21A ist eine Beispiel-Unterroutine. **Fig.** 20A ist ihr eX-Graph, das heißt eine Basis, auf welcher das eX-Modell (**8** in **Fig.** 7) gebaut wird. **Fig.** 20B ist ein eX-Verwahrungsort (**10** in **Fig.** 7), der durch den zuvor beschriebenen Prozess **18** in **Fig.** 7 aus der beispielhaften Unterroutine extrahiert worden ist. **Fig.** 21B ist X-Quellcode, der aus dem eX-Graph und eX-Verwahrungsort von Beispiel in **Fig.** 21A durch den X-Quellgenerator (**17** in **Fig.** 7) aufgebaut worden ist. Die X-Quelle enthält Aufrufe an das eX-Schnittstellenmodul (**11** in **Fig.** 7), welche Aufrufe für jedes Prozessende (Definition F3) instrumentiert werden:

```
CALL eXI(6,2)
                                        CALL eXI(14,2)
                                        CALL eXI(24,2)
                                        CALL eXI(31,2)
                                        CALL eXI(35,2)

vor jedem Systemeingabesignal (Ei von einem Ende):   CALL eXI(21,2)
und nach jedem sysout-Signal (Wi):                   CALL eXI(20,2)
```

wobei 6, 14, 24, 31 und 25 idt der Enden C2,*1,C1,= und R entsprechend sind, 20 die idw von W1 ist, 21 die ide von E1 ist und 2 die Modell-ID m ist, das in unserem Beispiel gleich 2 ist.
[0397] Die Modell ID m ist eine in Einser-Inkrement-Reihenfolge jedem unabhängigen Zielprozess (Hauptprozedur, Funktion oder Subroutine) zugewiesene Ganzzahl während des Extrahierens seines Modells und Verwahrungsort durch den Prozess **18**, **Fig.** 7.
[0398] Die erwähnten {idt,m} und {idw,m} und {ide,m} sind Synchronisierungsparameter **4** in **Fig.** 8.
[0399] In **Fig.** 20 sehen wir, dass die BA des Rückspulendeneintritts /1(1010) einen höheren Wert hat als die BA des entsprechenden Rücksprungeintritts *1(1001). Dies widerspricht noch nicht früher definierten Prinzipien, dass Rückspulenden die Steuerung dem Prozesselement rückgeben, das bereits in einem Prozesssegment mit kleinerer Binäradresse repräsentiert ist, da die Steuerung hier wirklich zum Eintritt einer DO-Schleife (Element D1) mit kleinerer Binäradresse zurückgegeben werden soll. Obwohl gemäß den Regeln einer DO-Schleifenimplementierung innerhalb einer Programmiersprache diese Rückgabe der Steuerung durch den Zwischenschritt des Durchlaufens durch das DO-Ende (Element C1) gemacht werden muss, das wiederum die Steuerung an D1 zurückgibt.

8 "Psuedo-parallele" Ausführung des Zielprozessmodells: "verzögert sequentielle" und "verzögert parallele" eX-Modell-Ausführung;

[0400] Durch Ausführung des eX-Modells des Zielprozesses verstehen wir den Prozess des Verfolgens seines eX-Graphen gemäß dem Ausführungspfad des Zielprozesses;

Verzögerte sequentielle Ausführung des eX-Modells:

[0401] "Verzögert sequentielle Ausführung des eX-Modells" wird durch CPU(s) implementiert, d. h. eine oder mehrere zwischen zwei Aufgaben umschaltende CPUs, wo eine Aufgabe das "Ausführungssegment" des Zielprozesses ist und die andere Aufgabe die Modulierung der ersten Aufgabe im eX-Modell dieses Prozesses durch die eX-Analysemaschine ist, die das "Ausführungssegment" als Teil des zwischen den folgenden Elementen von eX-Objektsatz lokalisierten Zielprozesses definiert aufweisen: Prozesseintritt, Punkte der Systemeingabe (Synchronisierung {ide,m} und "Prozessenden" (Synchronisierung {idt,m});

Verzögert parallele Ausführung des eX-Modells:

"verzögert parallele Ausführung des eX-Modells" wird durch mehr als eine CPU implementiert, wobei die Aufgabe des "Ausführungssegments" des Zielprozess und die Aufgabe des Modellierens der zuvor ausgeführten

"Ausführungssegmente" durch die eX-Analysemaschine parallel durchgeführt werden;
beide Verfahren der "pseudo-parallelen" Ausführung des eX-Modells werden durch Synchronisierungssignale von Prozessenden {die,m} und Systemeingabe {ide,m} implementiert, aber im ersten Fall wird die Implementierung der eX-Analysemaschineprozesse in der Form von CALL Anweisungen durchgeführt, wo der Zielprozess darauf wartet, dass die Steuerung von eXI-Modul zurück kommt, und im zweiten Fall werden die parallelen Prozesse durch die Synchronisierungssignale aktiviert. Die Synchronisierungssignale {ide,m} und {idt,m} sind entworfen, um das eX-Modell mit dem Zielprozess "up to date" zu machen. Die Synchronisierungssignale {idw,m} sind entworfen, um die ISOPS-Karte mit dem Zielprozess und der Prozesszeit innerhalb des eX-Modells "up to date" zu machen.

[0402] Da Signale von Prozessenden eine minimale Überbauinstrumentierung repräsentieren, wie vorstehend diskutiert worden ist, repräsentieren sie auch die minimalen Überbausynchronisierungssignale, um das Modell mit dem Zielprozess up to date zu halten. Synchronisierung des Punkts der Systemeingabe ist bequem, da während der "Benutzerdenkzeit" der Zielprozess sowieso unterbrochen wird.

9 Laufenlassen der eX-Analysemaschine. Aufbauen und Verbreiten der analysierten Kalkulationsbasis von Ursache-Wirkungsbeziehung innerhalb des Zielprozesses; Wissensinduktion;

[0403] Bezugnahme auf **Fig.** 8. Die folgenden Prozesse treten auf, wenn eXI(11) ein Synchronisierungssignal empfängt, das in Form {id,m} oder {k,b,m} implementiert ist (**4** in **Fig.** 8). Wir nehmen die Implementierung in der Form {id,m} an. eXI übersetzt {id} in {kb} des entsprechenden Modells {m}, den Wert von MK des Modells m wissend durch die früher beschriebene "Formel der ID":

$b = (id - 1)/mk + 1$

$k = id - (b - 1)*mk$

[0404] Prozessadresse {k,b} zeigt entweder auf Prozessende (Definition F3), dann ist die {id} {idt} oder auf sysout-Signal (W), dann ist {id} gleich {idw}, oder auf eine Eingabe vom Ende (E), dann ist die id gleich {ide}, keine andere Möglichkeit existiert;
[0405] Falls eXI den {idt} Synchronisationsparameter empfangen hat, wird dieser {idt} in die Systemausführungsgeschichtsdatei (**3** in **Fig.** 8) geschrieben.
[0406] Falls das Synchronisierungssignal {ide,m} oder {idt,m} ist, initiiert eXI die eX-Analysemaschine, und die folgenden Prozesse treten auf:

- die eX-Analysemaschine verfolgt das Modell vom Punkt des vorherigen Synchronisierungssignals {k,b} oder vom Eintrittspunkt zum {k,b} des derzeitigen Synchronisierungssignals zurück, wie zuvor beschrieben worden war, als wir die minimale Überbauinstrumentierung zum Aufzeichnen der Prozessausführungsgeschichte diskutiert haben.
- während des Durchlaufens jedes Elements des eX-Modells wird sein Lokalzeitattribut {p} um 1 inkrementiert und damit aktuell gehalten.
- Wenn das (Wi)Element getroffen wird, das eine Systemausgabe ist, initiiert die eX-Analysemaschine Zyklen von "Wissensinduktion"-Ketten (Anspruch 9) (siehe Definition K10: Wissensverbreitungsende"-Ereignis (KP-Ende) und **Fig.** 14, wo jeder Zyklus durch einen Systemausgabeparameter begonnen wird, falls (Wi) mehr als einen Ausgabeparameter hätte.

[0407] Wissensverbreitungsenden stabilisieren die "Wissensinduktions"-Verarbeitung, die anderenfalls eine längere und längere Kette eines sekundären Prozesses bei Vorwärtsausführung des Zielprozesses werden würde. Der korrekte Algorithmusraum wird potentiell verbreitet und W-Besitzer-Ereignisse von PA werden durch die Wisseninduktionsprozesse etabliert, siehe Definition K12:
"W-Besitzer" eines Ereignisses" und **Fig.** 14, **Fig.** 15, **Fig.** 16, **Fig.** 17 und die zuvor beschriebene "REGEL DER WISSENSVERBREITUNG".
[0408] Jedesmal, wenn ein Ausführungsereigniszustand durch den "Wissensinduktionsprozess" von "unsicher" zu "korrekt" aktualisiert wird, wird das entsprechende Bit in dem entsprechenden ACB-Konstrukt von 0 nach 1 aktualisiert, wodurch das Wissen der ACB vergrößert wird (siehe Definition von ACB und **Fig.** 19).
[0409] Während das {ide,m} Synchronisierungssignal eXI einen {k,b,p,m} Eintrag in die SIPS-Karte (**1** in **Fig.** 8) legt, wo {p} derzeitig durch dasselbe Synchronisierungssignal aktualisiert wird und {k,b} aus {ide} durch die zuvor beschriebene "Formel der 1D" erhalten wird.
[0410] Bezugnahme auf **Fig.** 21 B, Anweisung CALL eXI(21,2): Der {k,b,p,m} Eintrag wird wie {5,3, p,2} aussehen, wobei {p} das derzeitig aktualisierte Lokalzeitattribut ist, das eXI vom eX-Modell bekommen wird. Die folgende Anweisung: WRITE(M,002)CH wird in die Monitordatei (Einheit M) (**19** in **Fig.** 7) die tatsächliche Ein-

gabeaufzeichnung legen. Sie wird von der vorübergehenden Monitordatei (M) in die Systemeingabegeschichtsdatei neben dem Synchronisierungsparameter {k,b,p,m} durch eXI während des nächsten Synchronisierungssignals kopiert werden.

[0411] Falls das Synchroniserungssignal vom Systemausgabeelement {idw,m} kam, initiiert eXI den zuvor beschriebenen Prozess, um die ISOPS-Karte zu aktualisieren und gibt dann die Steuerung an den Zielprozess zurück. Während {k,b,p,m}-Parameter der ISOPS-Karte erzeugt werden, nimmt eXI Lokalzeitparameter {p} vom entsprechenden Wi-Element des eX-Modells und inkrementiert ihn um 1, da das {idw,m} Signal nicht den Modellsynchronisierungsprozess der eX-Analysemaschine durchführt.

10 Aktivieren der eX-Analysemaschine "Wissensdeduktions"-Prozesse durch Referenz auf die ISOPS-Karte, um den Bereich von Fehlerpositionen oder Platz für Modifikationen mit konstant abnehmender Unsicherheit zu lokalisieren;

[0412] Die Fähigkeit, absolut jedes Ereignis innerhalb des Systems von Programmen von der Systemausgabe zu adressieren, wird durch jene vier Koordinaten {k,b,p,m} sichergestellt. In der Lage zu sein, jegliches Element innerhalb des Systems von der Systemausgabe – etwas das jedem Programmanwender oder Programmierer ohne Bezugnahme auf den Programmquellcode sichtbar ist, zu greifen und zu analysieren, repräsentiert das Originelle an der Schnittstelle unserer Erfindung, zwischen dem Anwender des analytischen Werkzeugs und dem Zielprozess.

[0413] "Wissensdeduktion": Bezugnahme auf **Fig.** 13: Der unsichere Algorithmusraum wird konstant durch die zuvor beschriebenen "Wisseninduktions"-Prozesse der eX-Analysemaschine minimiert. "Wissensdeduktions"-Prozesse werden wie folgt implementiert:

[0414] Ein Anwender der eX-Maschine betrachtet die Systemausgabe, die aus der Systemergebnisgeschichte durch den inkrementellen Bildextraktor/Konstruktor wieder erzeugt wird. Durch Positionieren eines Cursors, einer Maus oder einer anderen Zeigevorrichtung am Systemergebnis werden die entsprechenden {k,b,p,m}-Parameter der ISOPS-Karte aktiviert.

[0415] Falls das referenzierte Systemausgabeergebnis für die eX-Maschine als "unerwartet" und daher inkorrekt definiert ist, startet die eX-Analysemaschine "Wissendeduktions"-Prozesse, die denselben Regeln der "Wisseninduktion" des Verfolgens von Ereignissen in der zur Zielprozessausführung entgegengesetzten Richtung bis zu "Wissenverbreitungsenden" folgen, die in diesem Fall "Wissensdeduktionsenden" genannt werden. Der einzige Unterschied ist, dass kein Anwachsen des Wissens an den ACB-Konstrukten während des Prozesses des Wiedergewinnens des "unsicheren Algorithmusraums" durchgeführt wird, der ein Teil "Ereignisdefinitionsraums" des fraglichen Ausgabeereignisses ist. Der unsichere Algorithmusraum ist eine Kombination von UPA-Adressen auf den Ketten zu den "Wissensdeduktionsenden". In **Fig.** 13 ist er symbolisch durch //Raum innerhalb des d(e3)-Ereignisdefinitionsraums zur Prozesszeit t3 hergestellt. Wie in **Fig.** 13 zu sehen, wird der unsichere Bereich des Ereignisdefinitionsraums konstant minimiert bei konstantem Algorithmusraum des Zel-Prozesses und konstantem Vergrößern des korrekten Algorithmusraums.

[0416] Das W-Besitzer-Attribut von CPA gestattet es, das Wissen über den Zielprozess während der Variationen bei Ausgabeergebnissakzeptanz aktuell zu halten.

[0417] Wenn eines der zuvor bestätigten W-Ereignisse aberkannt wird, fällt das Zielprozesswissen auf das durch die Position des aberkannten W-Besitzerereignisses in der Zielprozesszeit bestimmten Niveau. Wir nennen diesen Prozess eX-Maschine "Verlieren des Wissens". In **Fig.** 93 könnte das Verlieren des Wissens innerhalb des folgenden Scenarios verstanden werden:

1. Der Zielprozess wird bis zur Prozesszeit (t3) durchgeführt, was den korrekten Algorithmusraum (CPAs werden in dieser Figur durch +-Zeichen dargestellt) bis zu dem zur Prozesszeit (t3) gezeigten Niveau vergrößert.
2. Später wird ein Versagen (oder die Notwendigkeit für eine Ergebnismodifikation) mit Referenz auf das Ergebnis zur Prozesszeit (t2) begründet. Das Niveau des Wissens (korrekter Algorithmusraum) wird bis auf das zur Prozesszeit (t2) gezeigte Niveau sinken. Dies wird durch Verlieren des Zustands von CPA nach UPA für jede PA, für die ein W-Besitzereignis nach Prozesszeit (t2) aufgetreten ist, durchgeführt.

[0418] Die Fähigkeit "Wissen zu verlieren" macht die eX-Maschinenprozesse flexibel und realistisch gemäß der Definition von Versagen, die relativ zu individuellen Erwartungen oder relativ zu veränderten Erwartungen eines Individuums oder den veränderten Systemspezifikationen ist.

11 Interpretieren von eX-Graph und eX-Verwahrungsort;

[0419] Bezugnahme auf **Fig.** 20: Die Steuerung des Zielprozesses wird stets voll durch den eX-Graphen repräsentiert, der eine zweidimensionale Basis ist, auf der das eX-Modell des Zielprozesses aufgebaut wird. Der Zielprozess kann durch Interpretieren von Anweisungen des eX-Verwahrungsortes ausgeführt werden, wäh-

rend das eX-Modell durch Verfolgen seines eX-Graphen (**Fig.** 18C) ausgeführt wird. Bei dieser Implementierung werden die folgenden Komponenten der in **Fig.** 7 dargestellten traditionellen Objektcodeimplementierung nicht benötigt:

– X-Quellgenerator;
– X-Quellcode;
– kompiliere und linke;
– X-Objektcode

[0420] Bezugnahme auf **Fig.** 18C: Ein Sprachinterpreter (**23**) interpretiert eX-Verwahrungsortanweisungen, die der eX-Treiber (**20**) aus dem eX-Verwahrungsort herauszieht, während das eX-Modell ausgeführt (geparst) wird.

[0421] Signal (**24**) vom Interpreter gibt einen Code zurück, der einer von drei Werten ist:

(a) eX-Verwahrungsortanweisung, die nicht eine logische Anweisung ist (nicht L oder D), wird ausgeführt;
(b) eX-Verwahrungsortanweisung, die eine logische Anweisung (L oder D) ist, wird negativ ausgeführt;
(c) eX-Verwahrungsortanweisung ist eine logische Anweisung (L oder D), die positiv ausgeführt wird;

[0422] Der eX-Treiber ergreift eine Aktion gemäß dem beim Parsen (Ausführen) des eX-Modells zurückgegebenen Code.

[0423] Bezugnahme auf **Fig.** 20 und **Fig.** 21: Der eX-Treiber produziert Synchronisierungssignale für das eXI-Modul: {idt,m}, {ide,m} und {idw,m} während der Ausführung des eX-Modells. Das eXI-Modul wird wiederum die eX-Analysemaschine in derselben Weise aktivieren, wie es während der pseudo-parallelen eX-Modellausführung gemacht wird. Der Unterschied liegt darin, dass während der pseudo-parallelen eX-Modellausführung dem "Ausführungssegment" der Prozess des Rückverfolgens desselben Segments im eX-Modell folgt. Während der Ausführung des Zielprozesses durch den eX-Treiber wird im Gegensatz dazu die Ausführung des Zielprozesses und die Ausführung seines eX-Modells tatsächlich durch denselben Prozess implementiert.

[0424] Unter Bezugnahme auf **Fig.** 20A könnten die folgenden Signale im {id,m}-Format zu verschiedenen Prozesszeiten erzeugt werden:

CALL eXI(6,2)
CALL eXI(14,2)
CALL eXI(20,2)
CALL eXI(21,2)
CALL eXI(24,2)
CALL eXI(31,2)
CALL eXI(35,2)

oder dasselbe im {k,b,m}-Format:

CALL eXI(6,1,2)
CALL eXI(6,2,2)
CALL eXI(4,3,2)
CALL eXI(5,3,2)
CALL eXI(8,3,2)
CALL eXI(7,4,2)
CALL eXI(3,5,2)

[0425] Wie im Falle jeglichen interpretierten Prozesses werden die Modifikationen am Zielprozess mit diesem Verfahren leichter durchgeführt, da keine Neukompilierung oder Neuverknüpfung notwendig ist. Zusätzlich können die Modifikationen an der Zielprozesslogik und an den ausführbaren Zielprozess-Komponenten hier separat vorgenommen werden, welches ebenfalls die Aufgabe der Modifikation vereinfacht.

12 eX-Objektcode und seine Ausführung;
Ausführung des eX-Modells als ein Teil des Zielprozessobjektcodes;

[0426] Bezugnahme auf **Fig.** 18A: eX-Objektcode besteht aus zwei Komponenten:

1. einer durch den eX-Graphen innerhalb des eX-Modells repräsentierten Steuerkomponente; und
2. einer durch in Maschinencode übersetzte eX-Verwahrungsortanweisungen repräsentierten ausführbaren Komponente;

[0427] Bei dieser Implementierung, wie im Falle des Interpretierens des eX-Graphen und eX-Verwahrungsortes werden die folgenden Komponenten der in **Fig.** 7 dargestellten Implementierung des traditionellen Objektcodes nicht benötigt:

– X-Quellgenerator;
– X-Quellcode;

– Kompiliere und Verknüpfe;
– X-Objektcode

[0428] Ausführung von eX-Objektcode besteht aus zwei Prozessen:
1. Der eX-Treiber (**20**) führt das eX-Modell aus, d. h. verfolgt den eX-Graphen;
2. ein Maschinencodefragment, das in Maschinencodeanweisung des eX-Verwahrungsorts vorübersetzt ist und keine Steueranweisungen enthält, wird durch den Prozess (**23**) ausgeführt, der zur Ausführung durch den eX-Treiber vorgelegt wird.

[0429] Wie im Falle des Interpretierens der eX-Verwahrungsortanweisung empfängt der eX-Treiber ein Signal als Ergebnis einer Codefragmentausführung und geht in die geeignete Richtung, wenn das Signal von der Logikanweisung (L- oder D-Element des eX-Objektsatzes) kam.
[0430] Der eX-Treiber erzeugt während der Ausführung des eX-Modells Synchronisierungssignale für das eXI-Modul: {idt,m}, {ide,m} und {idw,m}. Das eXI-Modul wird wiederum die eX-Analysemaschine in derselben Weise aktivieren, wie es während der pseudo-parallelen eX-Modellausführung gemacht wird.
[0431] Der Unterschied liegt darin, dass bei der pseudo-parallelen eX-Modellausführung dem "Ausführungssegment" das Rückverfolgen desselben Segments im eX-Modell folgte. Während der Ausführung des Zielprozesses durch den eX-Treiber werden im Gegensatz dazu die Ausführung des Zielprozesses und die Ausführung seines eX-Modells tatsächlich durch denselben Prozess implementiert.
[0432] Unter Bezugnahme auf **Fig.** 20A könnten die folgenden Signale im {id,m}-Format zu verschiedenen Prozesszeiten erzeugt werden:
CALL eXI(6,2)
CALL eXI(14,2)
CALL eXI(20,2)
CALL eXI(21,2)
CALL eXI(24,2)
CALL eXI(31,2)
CALL eXI(35,2)
oder das Gleiche im {k,b,m}-Format:
CALL eXI(6,1,2)
CALL eXI(6,2,2)
CALL eXI(4,3,2)
CALL eXI(5,3,2)
CALL eXI(8,3,2)
CALL eXI/7,4,2)
CALL eXI(3,5,2)
[0433] Bezugnahme auf **Fig.** 18B:
Ein ausführbares Komponentenadress-(ECA)-Attribut wird jedem ausführbaren Element PA des eX-Modells hinzugefügt. ECA ist der Versatz innerhalb der ausführbaren Komponentendatei des eX-Objektcodes (**22** in **Fig.** 18A) des Fragments vom Maschinencode, das der übersetzten eX-Verwahrungsortanweisung entspricht. Jedem Fragment folgt das Anweisungstrennbyte (SSB). Der eX-Treiber (**20** in **Fig.** 8) liest den Wert von ECA aus dem eX-Modell (**8** in **Fig.** 18A), liest das entsprechende Fragment vom Maschinencode und durchläuft es für die Ausführung durch das Modul (**23** in **Fig.** 18A), das die ausführbare Komponente von eX-Objektcode laufen lässt.
[0434] Als eines von verschiedenen Verfahren zum Überwachen der Systemeingabe wird das folgende implementiert: wenn Modul **21** in **Fig.** 18 die Eingabe von dem Systemende in ausführbare Komponenten des eX-Objektcodes übersetzt, fügt es eine nächste Anweisung hinzu, welches die Aktion einer Eingabe von einem Systemende zu einer Ausgabe in die Monitorpufferdatei (M) (**19** in **Fig.** 18) umkehrt.
[0435] Beispielsweise aus **Fig.** 21B: READ(*,002)CH wird gefolgt von WRITE(M,002)CH
[0436] Der eX-Treiber hat dann die Option, diese zweite Anweisung zu aktivieren oder zu deaktivieren, um die Überwachung der Systemeingabe zu steuern, während das nächste Codefragment, das dem Ausführungsprozess (**23**) zugeführt werden soll, ausgewählt wird.

13 Rekompilierung und Neuverknüpfung von eX-Objektcode;

[0437] Bezugnahme auf **Fig.** 18A und 18B.
[0438] Die eX-Objektcodearchitektur besteht aus zwei Komponenten: Steuerung und Ausführbares gestatten einfacheres Modifizieren des Zielprozesses. Wenn die Änderung einfach in der Steuerung liegt, wird die Steuerungskomponente verändert. Wenn die Änderung beim Modifizieren der ausführbaren Komponente liegt, wird ein neuer Eintrag in der ausführbaren Komponentendatei gemacht und die ECA-Referenz in der Steuerungs-

komponente wird aktualisiert.

[0439] Die Auflösung der Verknüpfung zwischen der Steuerung und der ausführbaren Komponente durch die ECA wird zur Laufzeit vorgenommen. Daher wird das Kompilieren, Rekompilieren und Interpretieren nur von neu erzeugten oder modifizierten Anweisungen von ausführbarer Komponente von eX-Objektcode notwendig; die Steuerungskomponente des eX-Objektcodes kann potentiell durch Einfügen, Löschen oder Modifizieren an dem, dem neu erzeugten eX-Verwahrungsortelement entsprechenden eX-Graphenelement rearrangiert werden.

[0440] In einigen Situationen, wenn nur die Steuerstruktur rearrangiert werden muss, kann eine Modifikation am eX-Objektcode gemacht werden, ohne irgendwelche Änderungen an der ausführbaren Komponente und durch einfaches Rearrangieren der Anordnung der Elemente innerhalb des eX-Graphen.

[0441] Eine Modifikation am eX-Objektcode kann in einigen Situationen durch Deaktivieren des Steuerkomponentenelements und damit Umgehen des entsprechenden Elements der ausführbaren Komponente gemacht werden.

[0442] Wenn die Auflösung der Verknüpfungen zwischen den Steuerungs- und ausführbaren eX-Objektcodekomponenten zur Laufzeit implementiert wird, muss Verknüpfen oder Wiederverknüpfen einer Gruppe von zwei oder mehr Zielprozessen, die durch eX-Objektcode implementiert sind, durch Verknüpfen nur der Steuerungskomponenten des eX-Objektcodes durchgeführt werden.

[0443] Da die eX-Maschine durch ihre Hochniveauschnittstelle jeglichen Elementarprozess, d. h. Anweisung durch Referenz auf das Systemergebnis und konsequente Analyse der eX-Analysemaschine, ansprechen kann, gibt es einen Weg zur automatischen Modifikation des Zielprozesscodes ohne traditionelle Verwendung eines Compillers zur Rekompilierung und eines Linkers zum Wiederverknüpfen. Die Steuerungskomponente kann automatisch durch Einfügung, Löschung oder Rearrangierung von Elementen innerhalb des eX-Graphen modifiziert werden, der durch einen Benutzer angewiesen wird, der auf die Systemausgabeereignisse mit einer Anforderung, ihre Anordnung zu ändern oder sie bedingt oder unbedingt zu aktivieren oder zu deaktivieren, zeigt. Die Bedingungen können aus existierenden Systemausgabeergebnissen mittels derselben Methode des Zeigens auf die existierende Systemausgabeergebnisse und konsequentes Analysieren durch die eX-Analysemaschine erhalten werden, welche die Geschichte von Ereignissen zurückverfolgt, wie es durch die "Wissendeduktions"-Methode gemacht wird.

[0444] Dies ist die Basis für das hier vorgeschlagene Verfahren des Resultatorientierten Programmierens (ROP), das eine Sprache sehr hohen Niveaus für die Vorwärtsentwicklung eines Computerprogramms ist. Tatsächlich wächst das Niveau dieses Verfahrens mit der Vorwärtsentwicklung des Zielprozesses, da es mehr und mehr Ereignisse zum Analysieren und Wiederverwenden gibt.

[0445] IF-externe Verweise auf aufgerufene Prozesse werden zur Laufzeit aufgelöst und keine Prozesse werden miteinander vorverlinkt, so das die DLL (dynamische Linkbibliothek) nur aus einem Prozess besteht, für welchen das eX-Modell und der eX-Verwahrungsort aufgebaut sind, dann wird keine Rekompilierung oder Neuverknüpfung bei Prozessmodifikationen notwendig, wenn:

(a) eine neue eX-Verwahrungsortanweisung hinzugefügt werden soll, sie von bereits existierenden für die derzeitigen Prozesse ausführbaren Komponentenelemente ausgewählt wird oder, falls sie nicht innerhalb der ausführbaren Komponente gefunden werden kann, sie interpretiert wird, wenn sie durch das entsprechende Element der Steuerungskomponente aktiviert wird;

(b) die Steuerungskomponente des eX-Objektcodes durch die eX-Maschine durch die zuvor beschriebene Weise des Modifizierens des eX-Graphen automatisch editiert wird.

14 Implementieren von eX-Maschinenprozessen auf allen Stufen des Testens und der Produktion;

[0446] Wir schlagen vor, die eX-Maschine auf jeder Stufe des Testens oder der Produktion zu verwenden, die durch eines der folgenden Verfahren implementiert werden:

eX-Maschinenprozesse werden gemeinsam mit der Ausführung von Zielcomputerprozessen ausgeführt, wobei die eX-Maschinenprozesse eine Kombination der folgenden Prozesse sind:

– "pseudo-parallel" zur Zielprozessausführung des eX-Modells innerhalb der traditionellen Objektcodeimplementierung, oder Ausführung des eX-Modells als eine Komponente des Zielprozessobjekts innerhalb der eX-Objektcodeimplementierung;

– "elementare Wissensinduktion", die durch die Systemausgabeereignisse durch Propagieren dieses Wissens zwischen den Zielprozesselementen bis zu den "Wissenverbreitungsenden" aktiviert wird;

– "Wissensdeduktions"-Prozesse, die den Bereich von Unsicherheit innerhalb des eX-Modells des Zielprozesses definieren;

15 Resultat-orientierte Programmierung (ROP)

[0447] Ein Verfahren der Resultat-orientierten Programmierung (ROP) gestattet es, zuvor erzeugte Prozesse

durch die Schnittstelle sehr hohen Niveaus und ohne manuelle Interaktion mit dem Prozessquellcode zu modifizieren oder darauf aufzubauen, der aufweist:

– eine Benutzerschnittstelle ist auf die Referenz auf die bestehenden Systemergebnisse durch die Fähigkeit, auf SYSOUT-Ereignisse zu referenzieren, beschränkt;

– Hinzufügungen oder Modifikationen werden automatisch durch eX-Maschinenprozesse durchgeführt, welche die Ereignisdefinitionsräume von referenzierten SYSOUT-Ereignissen verwenden;

16 Implementieren des Programmstethoskop;

[0448] Audioanalyse, kombiniert mit traditioneller Video(Graph)-Analyse gestattet ein besseres dynamisches Verständnis von Programmen oder Systemen von Programmen. Ausführung irgendeines Zielprozesses kann gesehen (gehört) werden als eine "Musik", die durch den Prozessalgorithmus definiert ist und eine Funktion der Eingabedatenkombination ist.

[0449] Während eines kognitiven Prozesses des Programmverstehens sind wir mit der Aufgabe des "Spielens auf dem Computer" konfrontiert. Dies ist definitionsgemäß ein dynamischer Prozess. Andererseits ist ein Graph (Bild) definitionsgemäß ein statisches Objekt. Der Unterschied zwischen Graph und Geräuschrepräsentierung eines dynamischen Prozesses kann durch Vergleichen zweier Arten von Wiedererschaffung von Musik verstanden werden: einer durch Lesen der Musiknoten (graphische Repräsentation) und ein anderer durch tatsächliches Spielen der Musik.

[0450] Wenn ein guter Musiker die Noten liest, hört er tatsächlich die Musik. Die graphische Repräsentation der Noten wird dann durch das Geräusch verstärkt, selbst wenn es nur vom Experten, der auf die Noten blickt, gesehen werden kann.

[0451] Die statische Natur des Zielprozessgraphen kann durch Zeigen des Flusses der Steuerung durch die Graphenstrukturelemente verändert werden. Dies kann durch Hervorheben eines Elementes implementiert werden, das eine Steuerung empfängt und somit durch Bewegen der letzten Hervorhebeposition innerhalb des Graphen. Audiosynchronisierung mit dem Prozess kann selbst als ein Prozessdiagnosemechanismus bleiben und sie kann mit der Videoeffektdiagnose kombiniert werden.

[0452] Ein Audioeftekt während einer Programmanalyse kann wie folgt erzielt werden:

1. Während paralleler oder nachträglicher Programmemulation durch Anwenden der nächsten aufgezeichneten Endadresse {k,b} auf das eX-Modell des Programms können wir die Programmausführung im eX-Graphen des Algorithmus des Programms verfolgen.

2. Das Videoverfolgen des Programmausführungsprozesses wird einfach durch Hervorheben des Elements des eX-Graphen erzielt, der während des Verfolgens aktuell aktiv ist.

3. Der Audioeffekt wird durch Aufrufen einer Hardware-internen Funktion hinzugefügt, die es gestattet, Audiofrequenzen zu erzeugen und sie zum Lautsprecher zu schicken. Ein Kopfhörer wird eine geeignetere Ergänzung für den Computer sein, wenn ein System durch das beschriebene Verfahren analysiert wird, um nicht andere Programmierer zu beeinträchtigen;

die Frequenz kann als Funktion von zwei Parametern berechnet werden:

(k,b), wo ein Einzelinkrement {k} die Frequenz um einen diskreten Schritt inkrementiert und ein einfaches Inkrement von {b} eine Frequenz um einen größeren Schritt inkrementiert, beispielsweise um eine Oktave. Eine Verzweigung zurück wird entsprechend die Frequenz dekrementieren.

[0453] Die Dauer des Beeps entsprechend dem Verfolgen eines individuellen Elements des eX-GRAPH kann so eingestellt werden, dass es vom menschlichen Ohr wahrgenommen werden würde.

[0454] Der Effekt einer solchen Audioausführungsemulation ist, dass jeder Fortschritt der Programmausführung seine eigene "Musik" hat, abhängig von der hartcodierten Struktur des Prozesses und der Eingabedatenkombination. Im Falle des Emulierens einer Schleife einer großen Zahl von Iterationen kann ein Codesignal gesetzt werden, um solch eine Schleife anzuzeigen, um die Notwendigkeit eines Verfolgens aller Iterationen zu eliminieren.

[0455] Der Effekt der Audioanalyse wird ein solcher sein, dass ein Anwender in der Lage sein wird, dem Programmverhalten zuzuhören, wie ein Doktor dem Patientenherzschlag zuhört.

1. Wenn ein bestimmtes Ausgabeergebnis über eine lange Zeit nicht beobachtet wird, kann diese Analyse aktiviert werden, um zu hören: – ist das System tot;

– ist das Programm in einer Schleife;

– wartet es auf eine Eingabe:

(wenn das letzte ausgeführte Element eine Eingabeanweisung war, kann ein spezieller Audioton erzeugt werden)

2. Die am stärksten belasteten Orte des Programmalgorithmus können als die üblichste Frequenz gehört und daher leicht identifiziert werden;

3. die Differenz im Programmverhalten bei unterschiedlichen Eingabedatenkombinationen kann gehört

werden;
4. ein spezieller Ton kann einen Aufruf einer Tochterfunktion anzeigen und ein spezieller Ton kann die Rückkehr von dem Aufruf anzeigen. Die Audioanalyse kann auf nur solche Signale ursprünglich beschränkt und optional expandiert werden, um Prozessausführung innerhalb einer spezifischen Funktion anzuzeigen, falls die Funktionalität in ihr studiert werden sollte.
5. Die Audioanalyse kann während der Prozessemulierung (direkt oder nachträglich) aufgezeichnet werden und dann ohne die tatsächliche Modusemulation in beiden Richtungen vorwärts und rückwärts abgespielt werden.
6. Sie kann synthetisiert mit dem Wiederabspielen des Systemergebnisses wieder abgespielt werden und damit ein Auffüllen der Intensität der für solche Ergebnisserzielungen notwendigen Verarbeitung geben;
7. Sie kann aufgezeichnet und für verschiedene Eingangsdatenkombinationen für denselben Prozess abgespielt werden, um den Unterschied in der Intensität von Berechnungen für diese unterschiedlichen Eingabesituationen zu analysieren.

[0456] Gegeben, dass das Potential für den Programmfehler eine Funktion der Anzahl der beteiligten Elementarprozesse ist, wird diese Analyse bei der Auswahl einer einfacheren (Eingabe-Prozess-Ausgabe)-Kombination während der Systementwicklung hin zu einer neuen Spezifikation helfen.

Patentansprüche

1. Vorrichtung zur automatischen Analyse eines Zielprozesses, wobei die Vorrichtung den Ort von im Zielprozess enthaltenen Fehlern oder den Ort von Modifikationen innerhalb des Zielprozesses aufgrund einer veränderten Ergebniserwartung bestimmt, wobei die Vorrichtung einen sekundären Prozess des Akkumulierens von Wissen zur Zielprozessfunktionalität durch Wissensinduktion erzeugt, umfassend:
ein Analyseberechnungsbasis-Attribut (ACB) für jedes ausführbare Element des Zielprozesses;
Mittel zum Ausführen des Zielprozesses und des sekundären Prozesses,
Mittel für den sekundären Prozess, um ein elementares Wissen zur Zielprozessfunktionalität durch Wissensinduktion aufzubauen und zu verbreiten, die das elementare Wissen durch einen Platz und eine Zeit innerhalb des Zielprozesses verbreiten,
wobei das elementare Wissen ein Wissen oder eine Unsicherheit über die Korrektheit eines Effekts des Zielprozesselementausführungsereignisses anzeigt, und
wobei das elementare Wissen innerhalb des ACB aufgezeichnet ist und
wobei der Ort des für ein spezifisches Versagen verantwortlichen Fehlers oder der Ort der Modifikation innerhalb des aufgrund veränderter Ergebniserwartung benötigten Zielprozesses durch Wissensdeduktion erhalten wird, die das durch die Wissensinduktion erhaltene elementare Wissen verwendet, um den Ort als einen fortwährend minimierten Bereich von Unsicherheit innerhalb des Ereignisdefinitionsraums des Ergebnisses zu definieren und die Wissensdeduktion denselben Regeln der Wissensinduktion des Verfolgens von Ereignissen in der zur Zielprozessausführung entgegengesetzten Richtung folgt.

2. Vorrichtung nach Anspruch 1, wobei der Zielprozess durch das Ausführen von Code in traditioneller Objekt-Code-Form implementiert ist, während Synchronisationssignale vom Zielprozess zum sekundären Prozess gesendet werden.

3. Vorrichtung nach Anspruch 1, wobei der Zielprozess durch einen Treiber implementiert ist, der ein Modell des Zielprozesses verfolgt und entsprechende ausführbare Elemente des Zielprozesses, die in ihrer Quellcode-Form repräsentiert sind, einem Sprachinterpreter präsentiert, während der sekundäre Prozess durch den Treiber aktiviert wird.

4. Vorrichtung nach Anspruch 1, wobei der Zielprozess durch einen Treiber implementiert ist, der ein Modell des Zielprozesses verfolgt, wobei solch ein Modell auf die Steuerelemente des Zielprozesses aufbaut, und entsprechende, in ihrer Maschinencodeform repräsentierte Nichtsteuerelemente des Zielprozesses dem Computerbetriebssystem präsentiert, während der sekundäre Prozess durch den Treiber aktiviert wird.

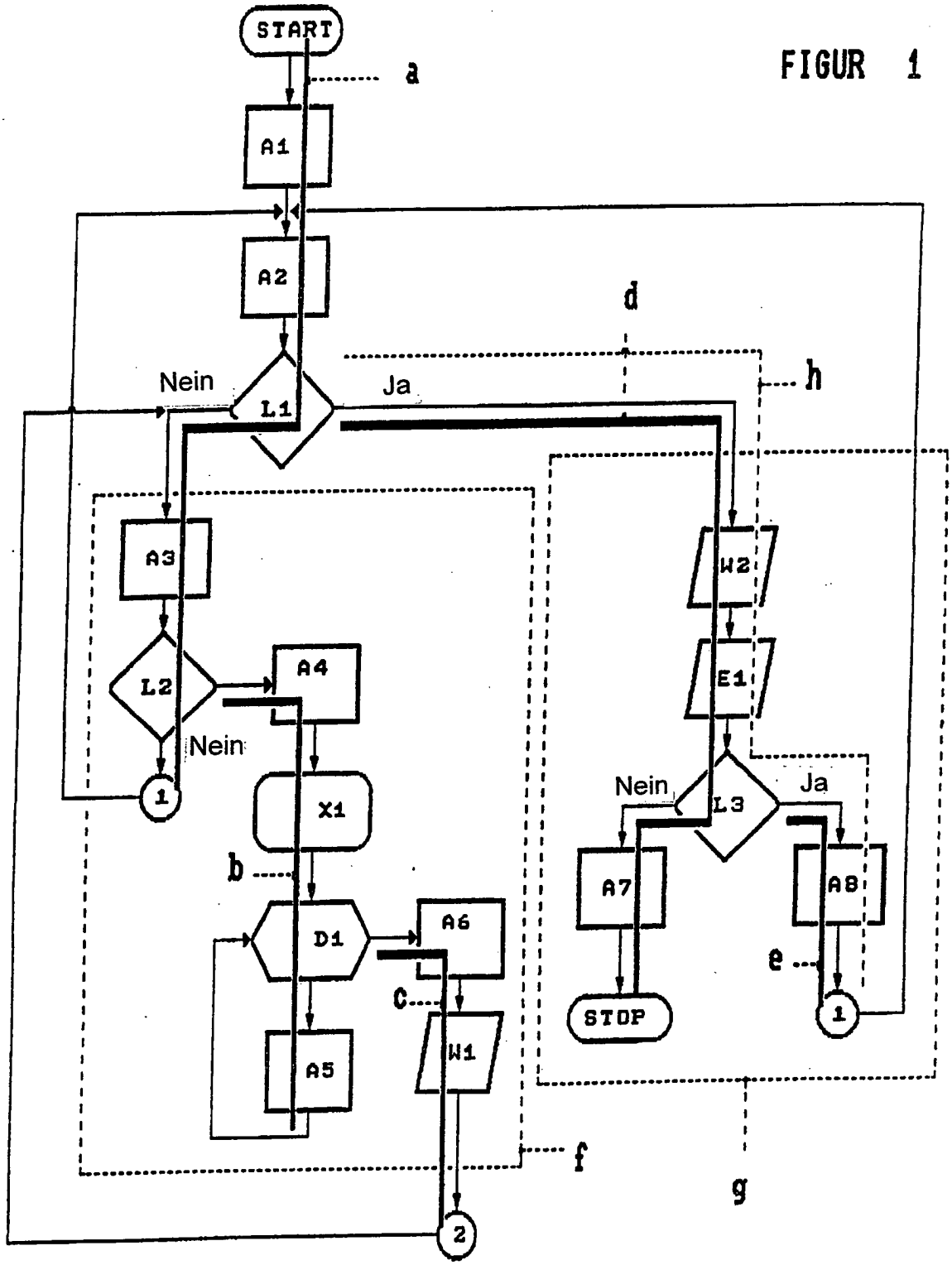
5. Vorrichtung nach Anspruch 1, weiterhin umfassend Mittel zum Konstruieren der Definitionen von Modifikationen innerhalb des Zielprozesses, die aufgrund der veränderten Ergebniserwartung benötigt werden, wobei die Definitionen durch Wiederverwenden der Ereignisdefinitionsräume der . vorigen Ergebnisse des Zielprozesses erhalten werden.

6. Vorrichtung zur automatischen Analyse eines Zielprozesses, umfassend:
Feststellen eines Fehlerorts, der im Zielprozess enthalten ist, oder eines Orts von Modifikationen innerhalb des

Zielprozesses, die aufgrund von veränderter Ergebniserwartung notwendig sind;
Erzeugen eine sekundären Prozesses des Akkumulierens von Wissen zur Zielprozessfunktionalität durch Wissensinduktion;
Etablieren eines Analyseberechnungsbasis-Attributs (ACB) für jedes ausführbare Element des Zielprozesses;
Ausführen des Zielprozesses und des sekundären Prozesses;
Ausführen des sekundären Prozesses, der ein elementares Wissen zur Zielprozessfunktionalität durch Wissensinduktion aufbaut und verbreitet, der das elementare Wissen durch einen Platz und eine Zeit innerhalb des Zielprozesses verbreitet, wobei das elementare Wissen ein Wissen oder eine Unsicherheit über die Korrektheit eines Effekts des Zielprozesselementausführungsereignisses anzeigt;
Aufzeichnen des elementaren Wissens innerhalb der ACB; und
Erhalten des für ein spezifischen Versagen verantwortlichen Fehlerorts oder des Modifikationsorts innerhalb des aufgrund, veränderter Ergebniserwartung benötigten Zielprozesses durch Wissensdeduktion, die das durch die Wissensinduktion erhaltene elementare Wissen verwendet, um den Ort als einen fortwährend minimierten Bereich von Unsicherheit innerhalb des Ereignisdefinitionsraums des Ergebnisses zu definieren und die Wissensdeduktion denselben Regeln der Wissensinduktion des Verfolgens von Ereignissen in der zur Zielprozessausführung entgegengesetzten Richtung folgt.

Es folgen 19 Blatt Zeichnungen

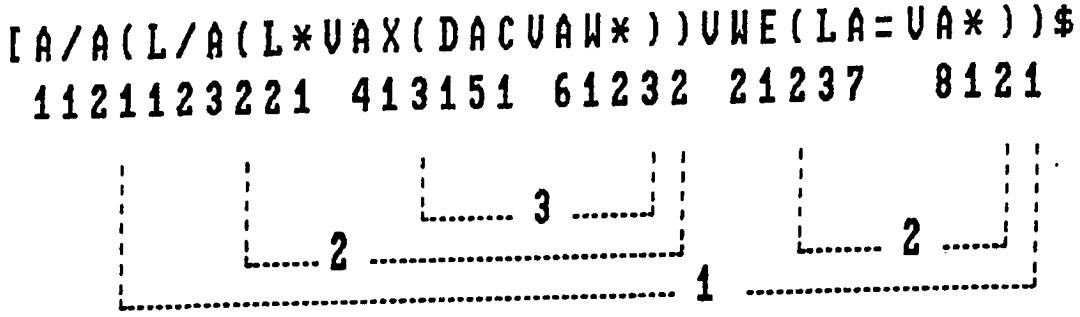
Anhängende Zeichnungen



FIGUR 1

1 / 19

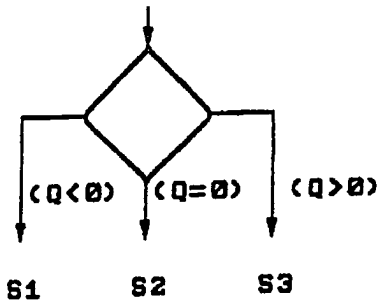
FIGUR 2


[A/A(L/A(L*VAX(DACVAW*))VWE(LA=VA*))$
1121123221 413151 61232 21237 8121
3
2
2
1

FIGUR 3

Beispiel:

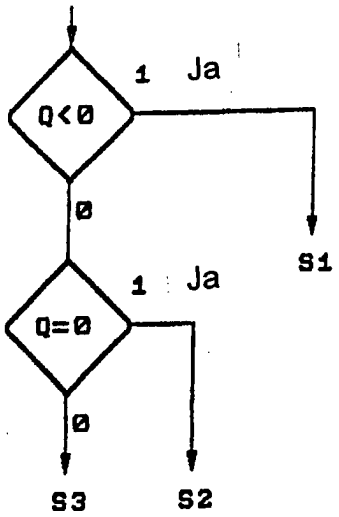
```
IF (Q) S1,S2,S3
```



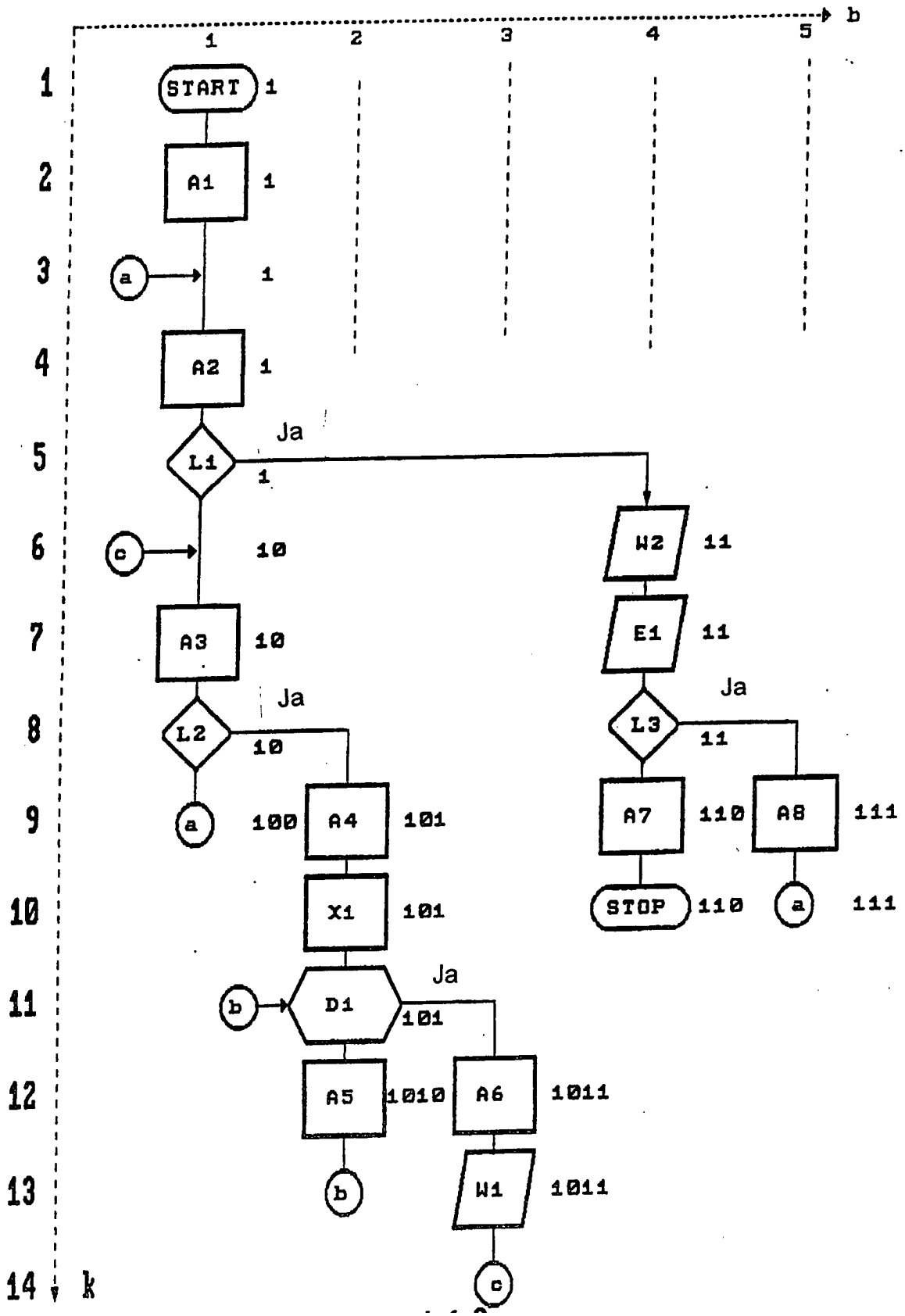
Kann reduziert werden auf:

```
IF(Q.LT.0) GO TO S1
IF(Q.EQ.0) GO TO S2
GO TO S3
```

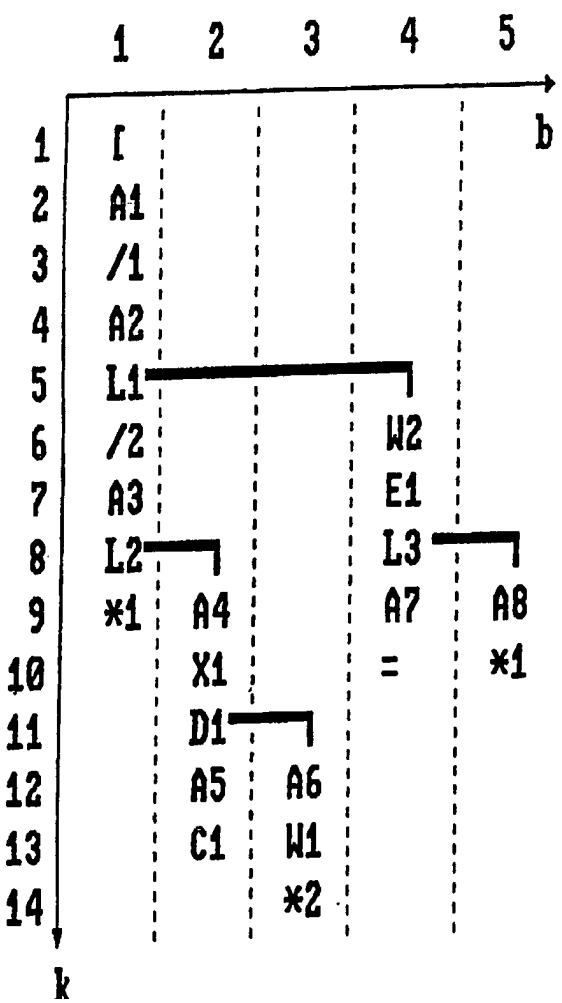
wird reduziert auf:



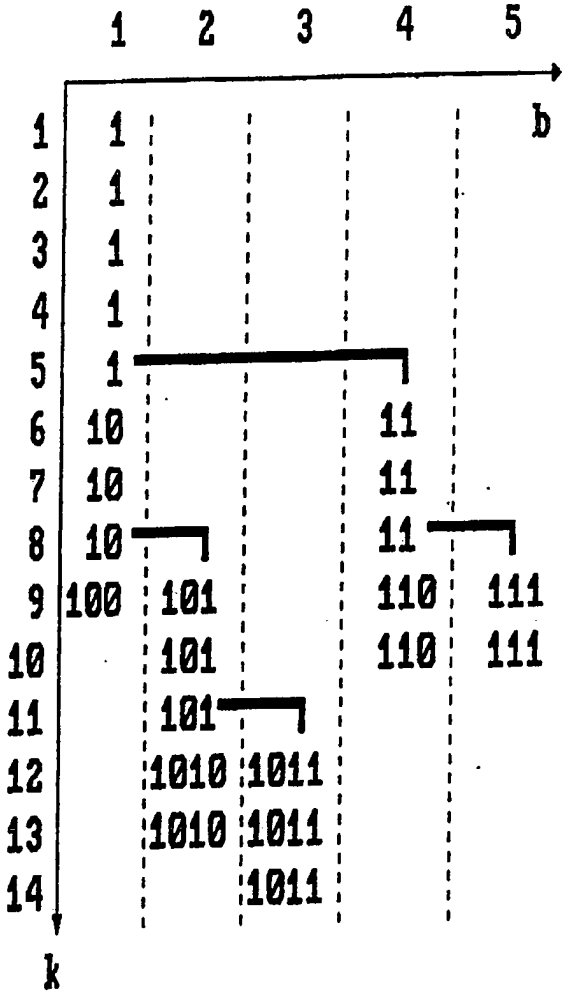
FIGUR 4


b
1
2
3
4
5
k
START
A1
a
A2
L1
Ja
c
A3
L2
Ja
a
A4
X1
b
D1
Ja
A5
A6
b
W1
c
W2
E1
L3
Ja
A7
A8
STOP
a
1
1
1
1
1
10
10
10
100
101
101
101
1010
1011
1011
11
11
11
110
111
110
111

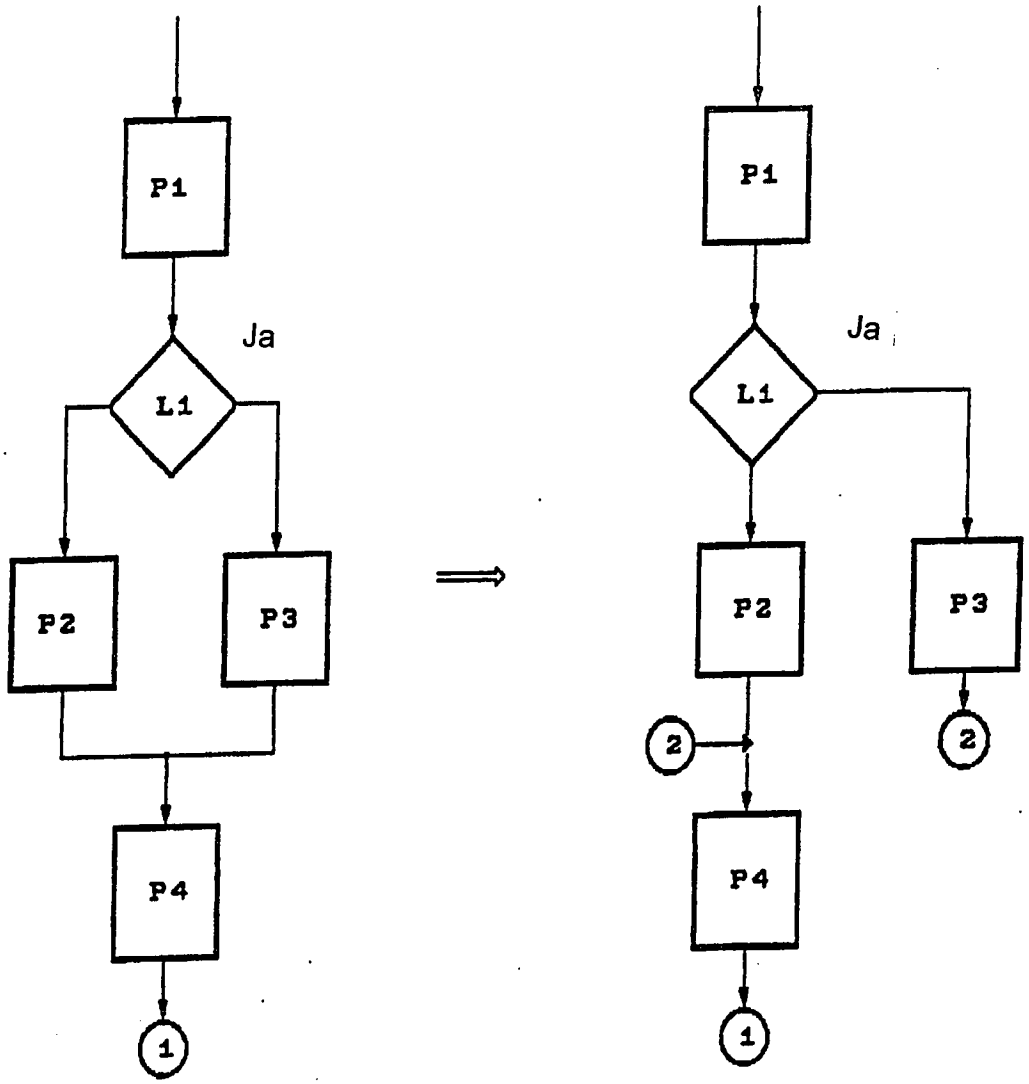
FIGUR 5 A



FIGUR 5 B



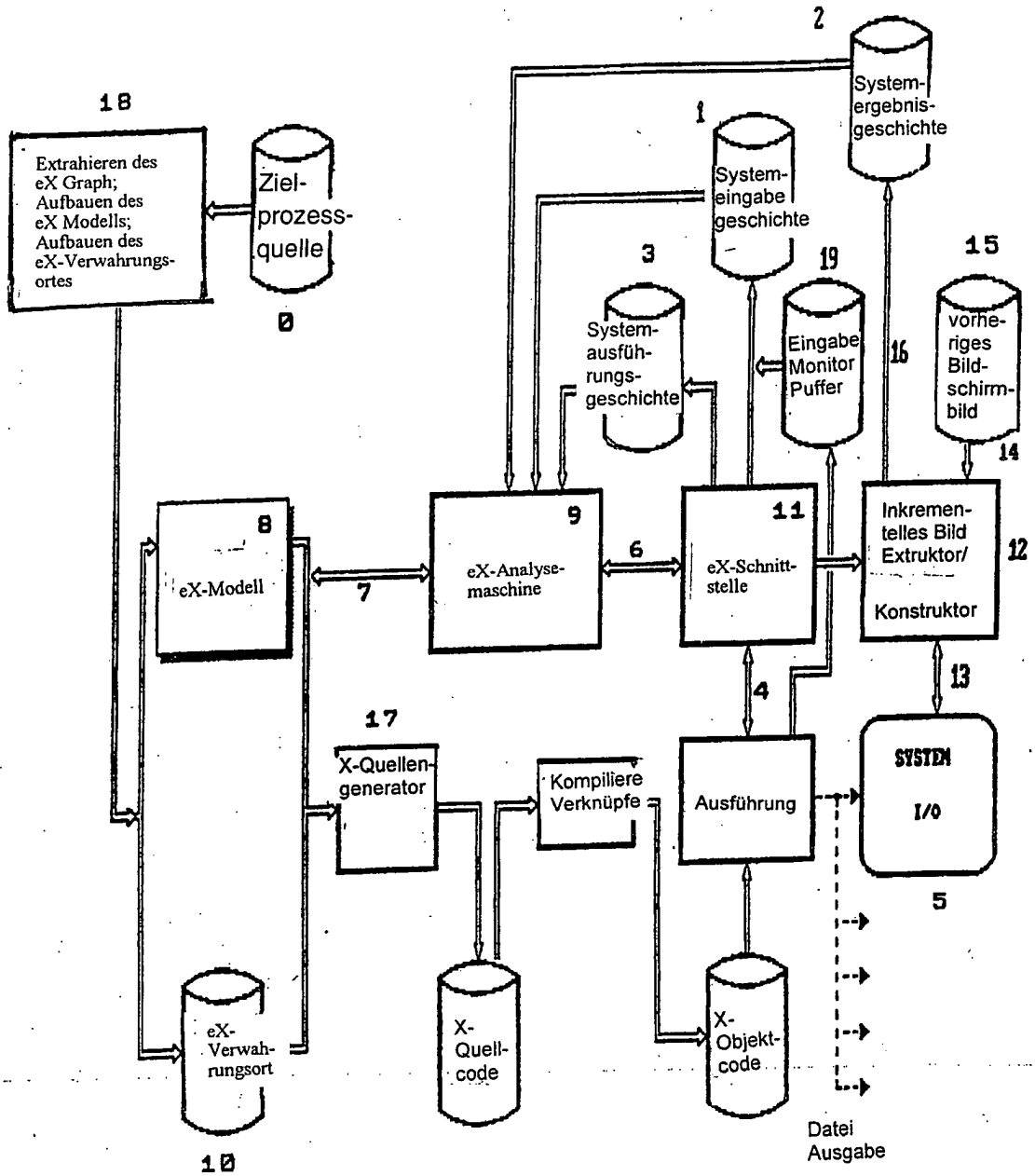
FIGUR 6

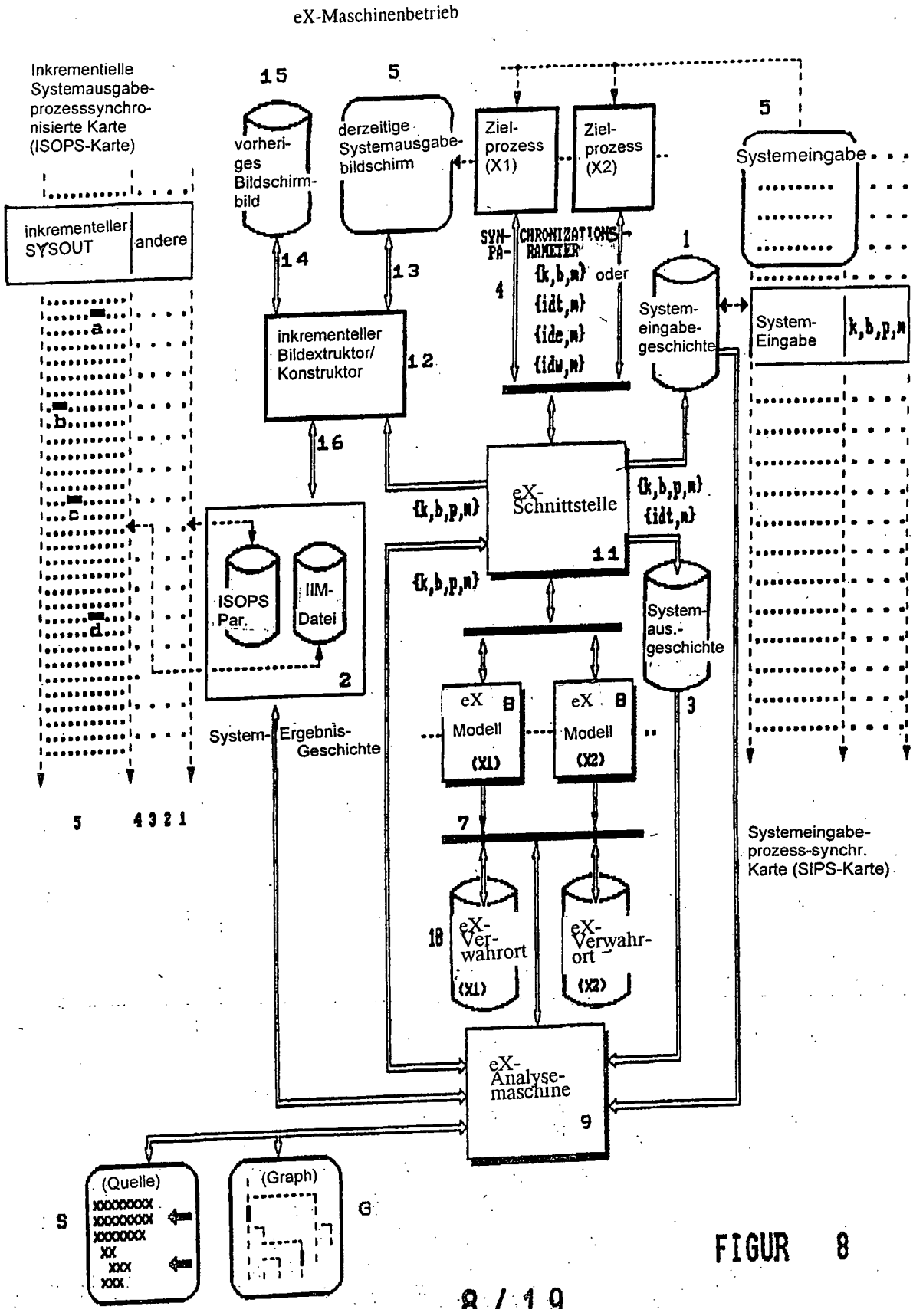



```
P1
IF (L1) THEN
  P3
ELSE
 P2
ENDIF
P4
```

```
P1 (L1 P2 !2 P4 *1 V P3 *2)
   i                    i
```

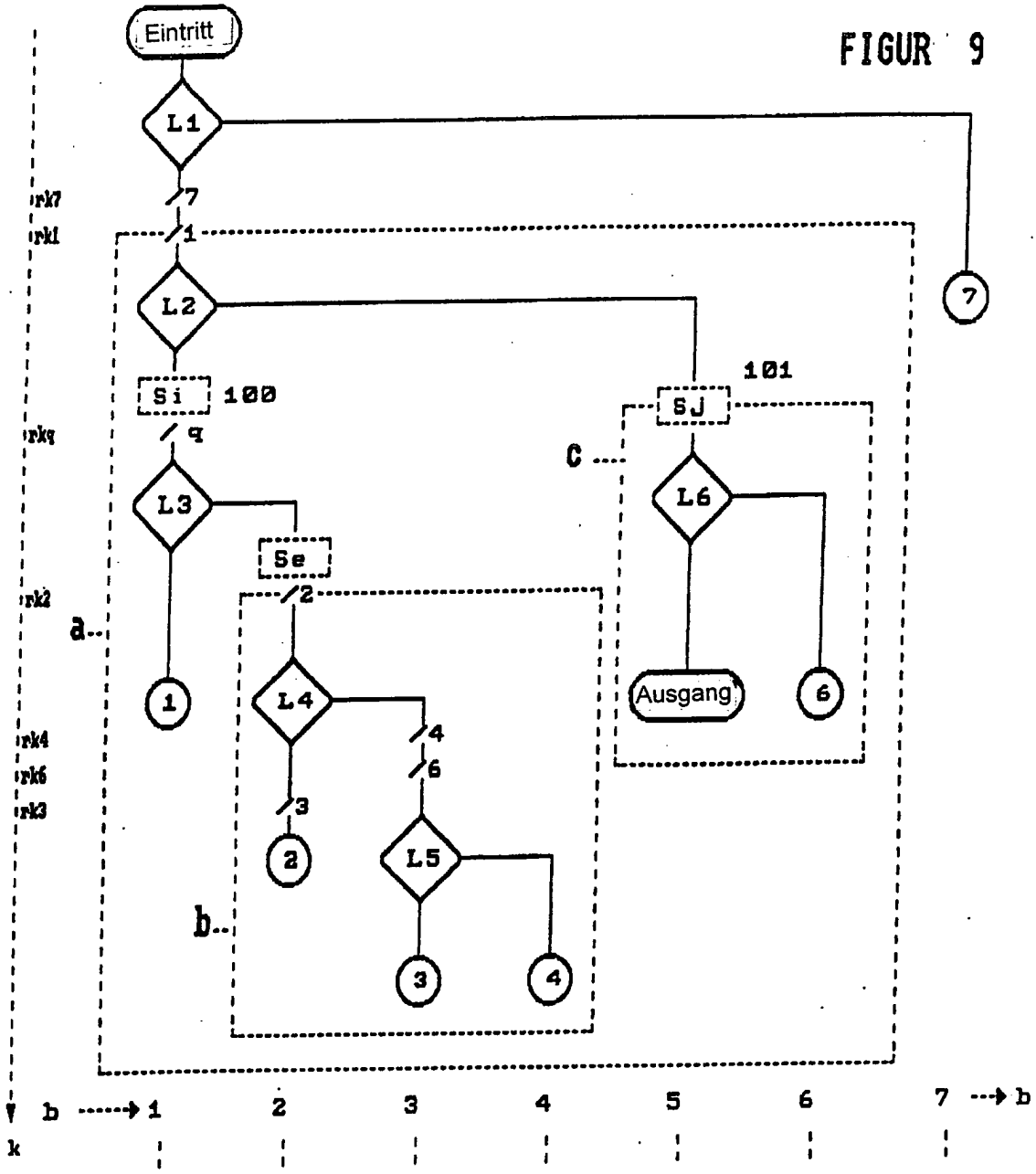
FIGUR 7


18
Extrahieren des eX Graph; Aufbauen des eX Modells; Aufbauen des eX-Verwahrungs-ortes
Ziel-prozess-quelle
0
2
System-ergebnis-geschichte
1
System-eingabe geschichte
3
System-ausführ-ungs-geschichte
19
Eingabe Monitor Puffer
16
15
vorhe-riges Bild-schirm-bild
14
8
eX-Modell
7
9
eX-Analyse-maschine
6
11
eX-Schnitt-stelle
Inkremen-telles Bild Extruktor/ Konstruktor
12
13
4
17
X-Quellen-generator
Kompiliere Verknüpfe
Ausführung
SYSTEM I/O
5
eX-Verwah-rungsort
10
X-Quell-code
X-Objekt-code
Datei Ausgabe


eX-Maschinenbetrieb
Inkrementielle Systemausgabe-prozesssynchro-nisierte Karte (ISOPS-Karte)
inkrementeller SYSOUT
andere
a
b
c
d
5 4 3 2 1
15
vorheriges Bildschirm-bild
5
derzeitige Systemausgabe-bildschirm
Ziel-prozess (X1)
Ziel-prozess (X2)
5
Systemeingabe
SYN-PA-
CHRONIZATIONS-RAMETER
{k,b,m}
{idt,m}
{ide,m}
{idu,m}
oder
4
1
System-eingabe-geschichte
System-Eingabe
k,b,p,m
14
13
inkrementeller Bildextruktor/ Konstruktor
12
16
{k,b,p,m}
eX-Schnittstelle
11
{k,b,p,m}
{idt,m}
{k,b,p,m}
ISOPS Par.
IIM-Datei
2
System-Ergebnis-Geschichte
System-aus.-geschichte
3
eX B Modell (X1)
eX B Modell (X2)
7
Systemeingabe-prozess-synchr. Karte (SIPS-Karte)
18
eX-Ver-wahrort (X1)
eX-Verwahr-ort (X2)
eX-Analyse-maschine
9
(Quelle)
S
(Graph)
G

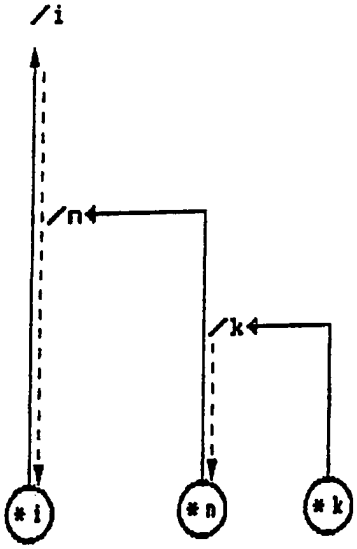
FIGUR 8

FIGUR 9

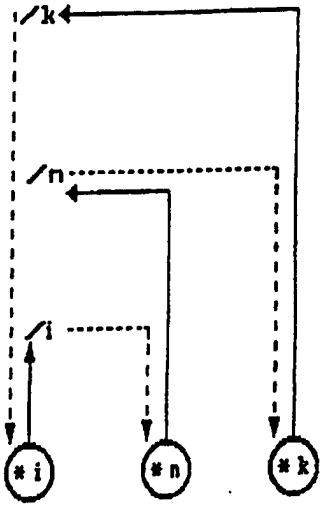


EXIT ACC. IND.	orig.	0	0	0	0	1	0	0
	end	1	0	0	0	1	0	1
PUP	rk orig.	rk1	rk2	rk3	rk4	0	rk6	rk7
	rk end	rk1	rk2	rk2	rk2	0	rk2	rk7
	BA orig.	10	1001	10010	10011	0	10011	10
	BA end	10	1001	1001	1001	0	1001	10
RBA		10	1001	10010	10011	0	10011	10
TBA		1000	10010	100110	100111	1010	1011	11

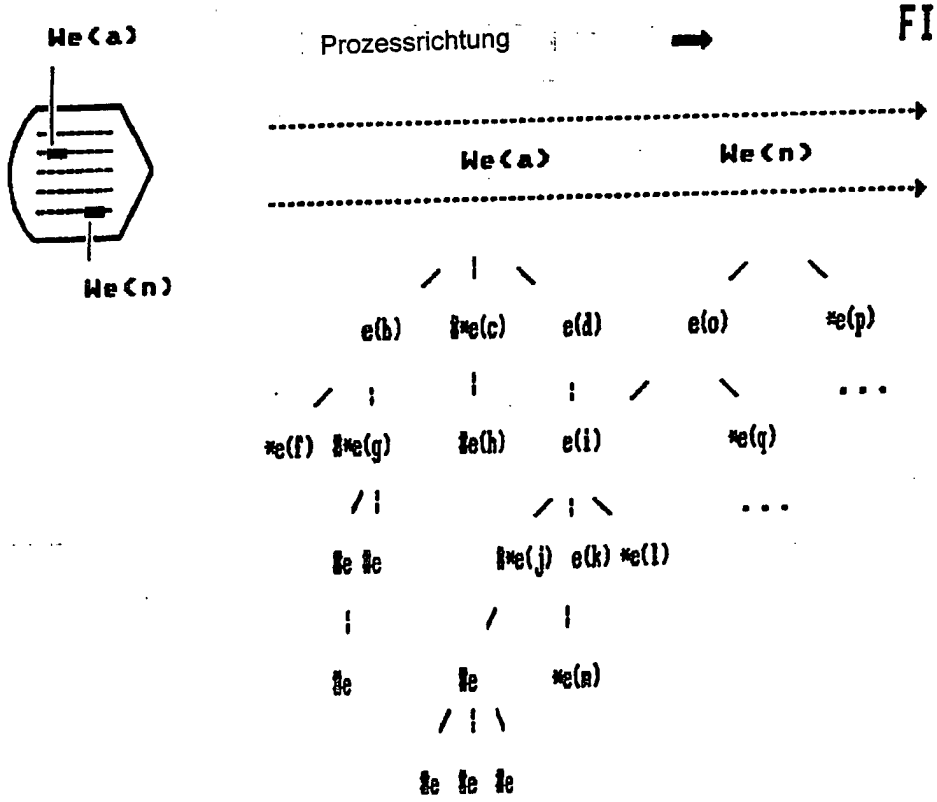
FIGUR 10 A



FIGUR 10 B

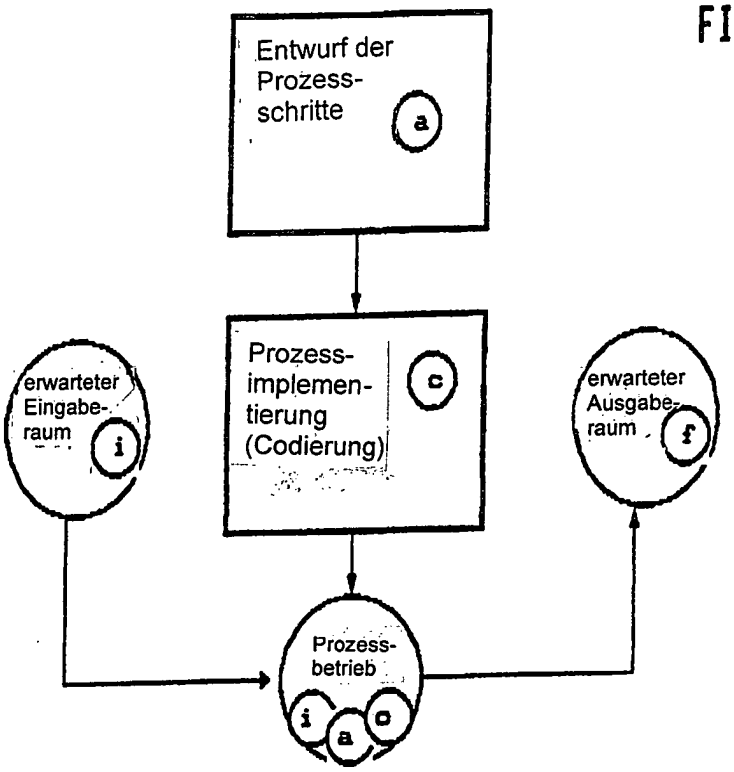


FIGUR 14

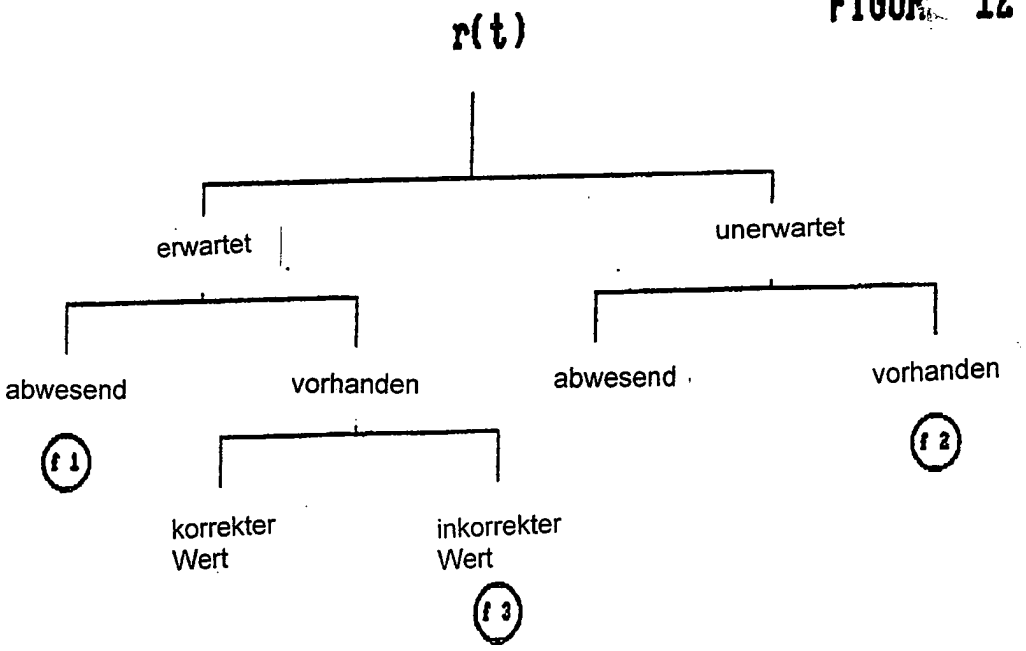


SUBSTITUTE SHEET (RULE 26)

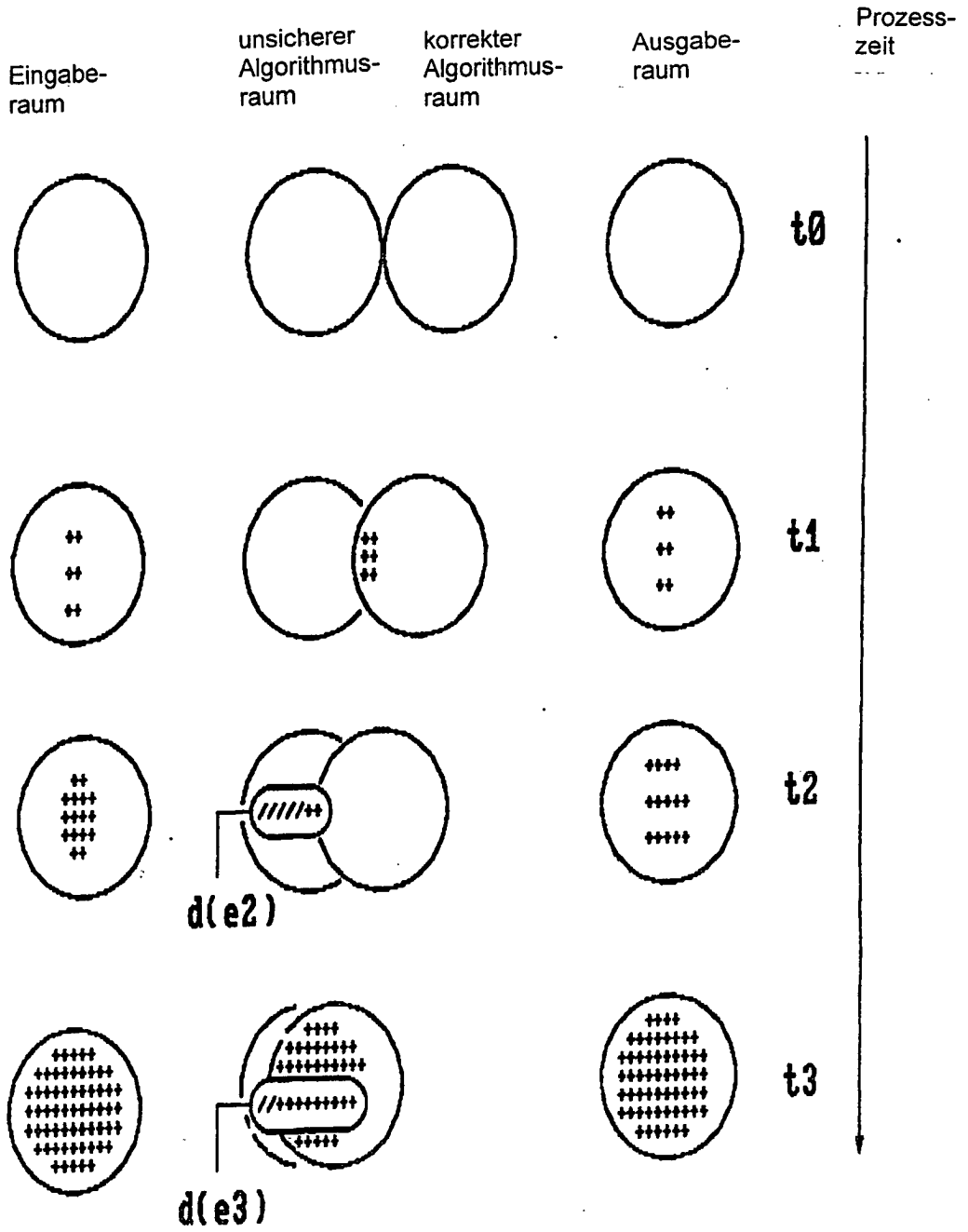
FIGUR 11



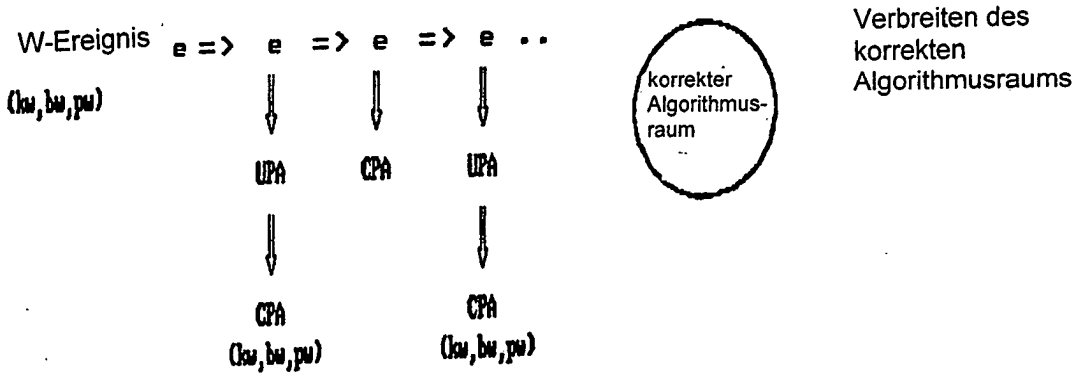
FIGUR 12



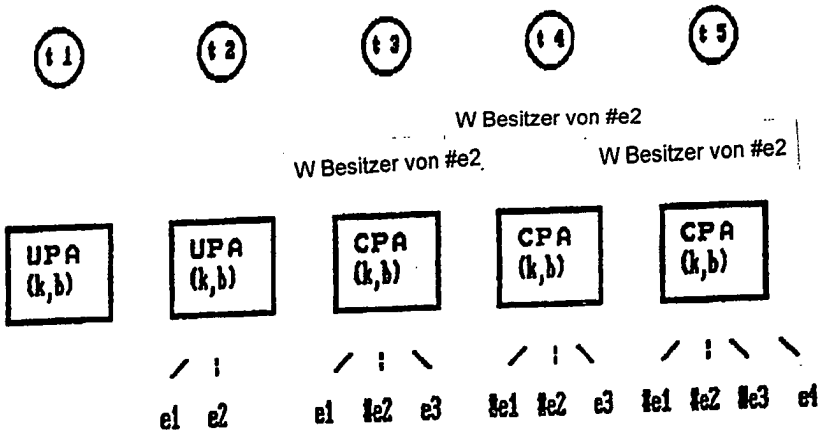
FIGUR 13


Eingabe-raum
unsicherer Algorithmus-raum
korrekter Algorithmus-raum
Ausgabe-raum
Prozess-zeit
t0
t1
t2
t3
d(e2)
d(e3)

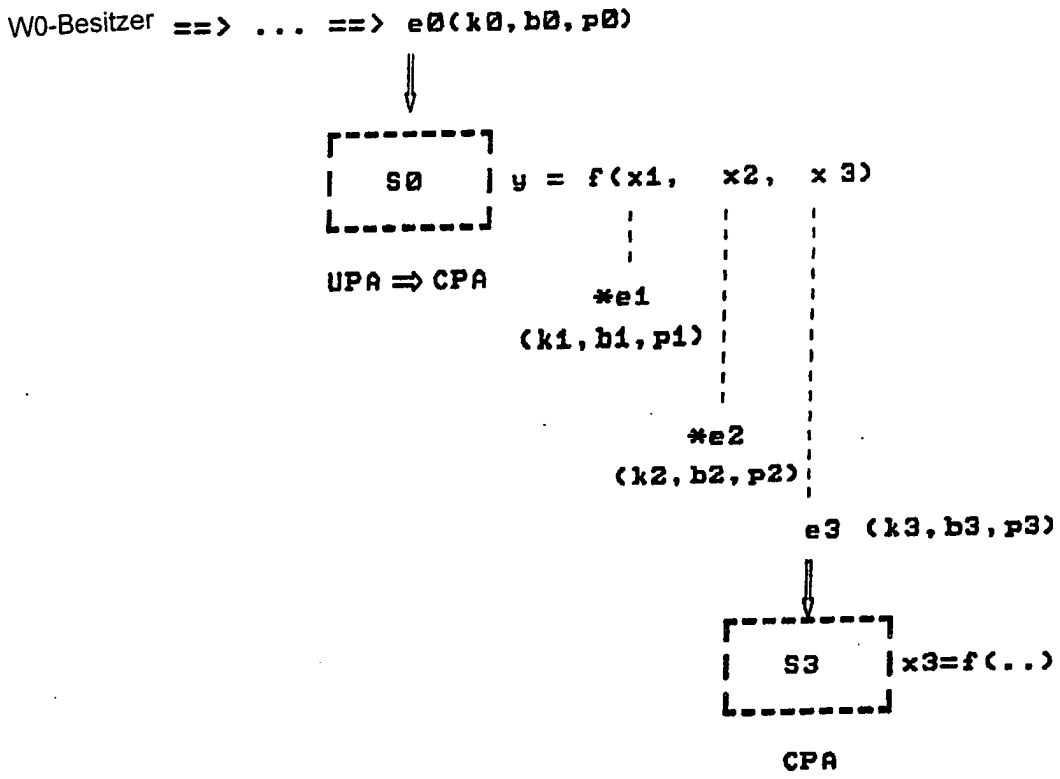
FIGUR 15

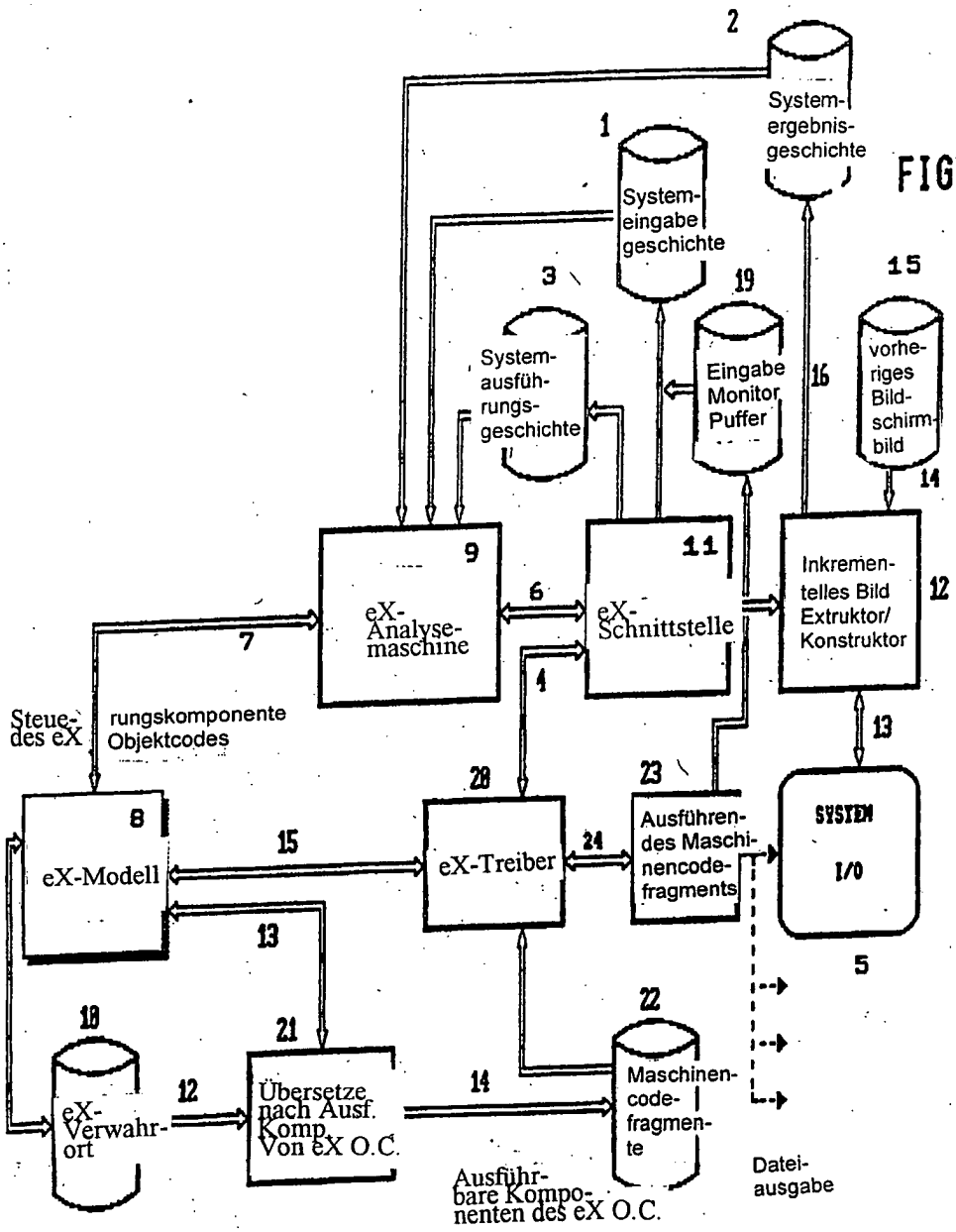


FIGUR 16

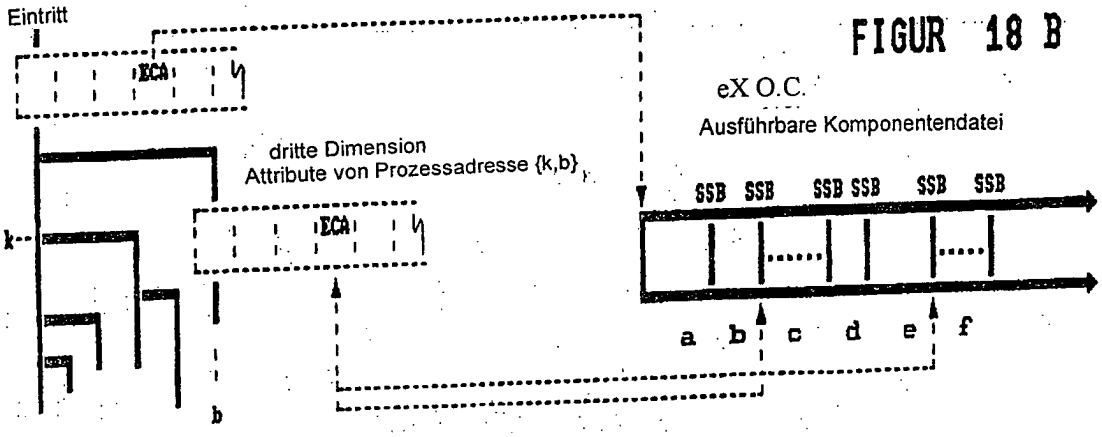


FIGUR 17

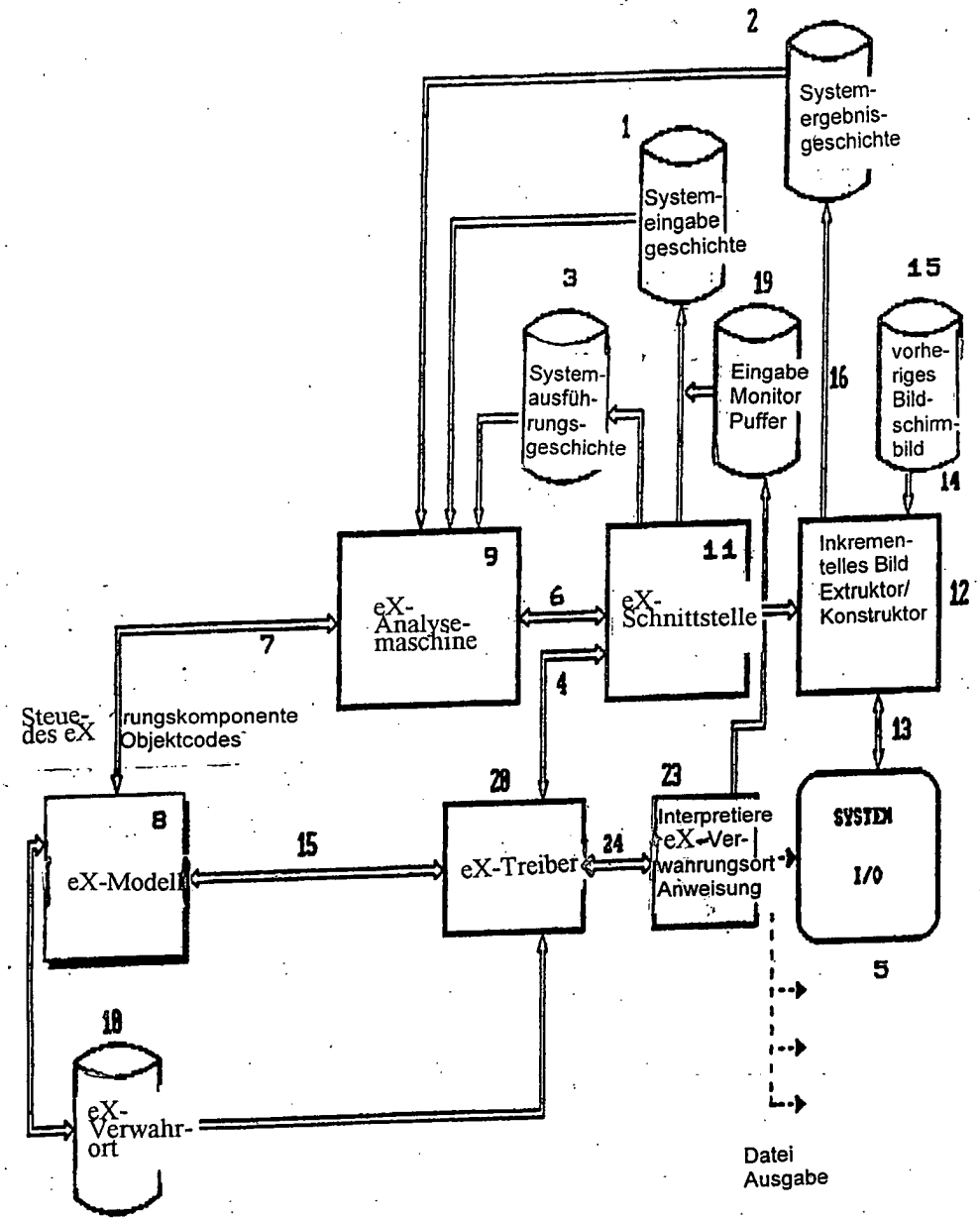

W0-Besitzer ==> ... ==> e0(k0,b0,p0)
S0
y = f(x1, x2, x 3)
UPA ⇒ CPA
*e1
(k1,b1,p1)
*e2
(k2,b2,p2)
e3 (k3,b3,p3)
S3
x3=f(..)
CPA



FIGUR 18 A

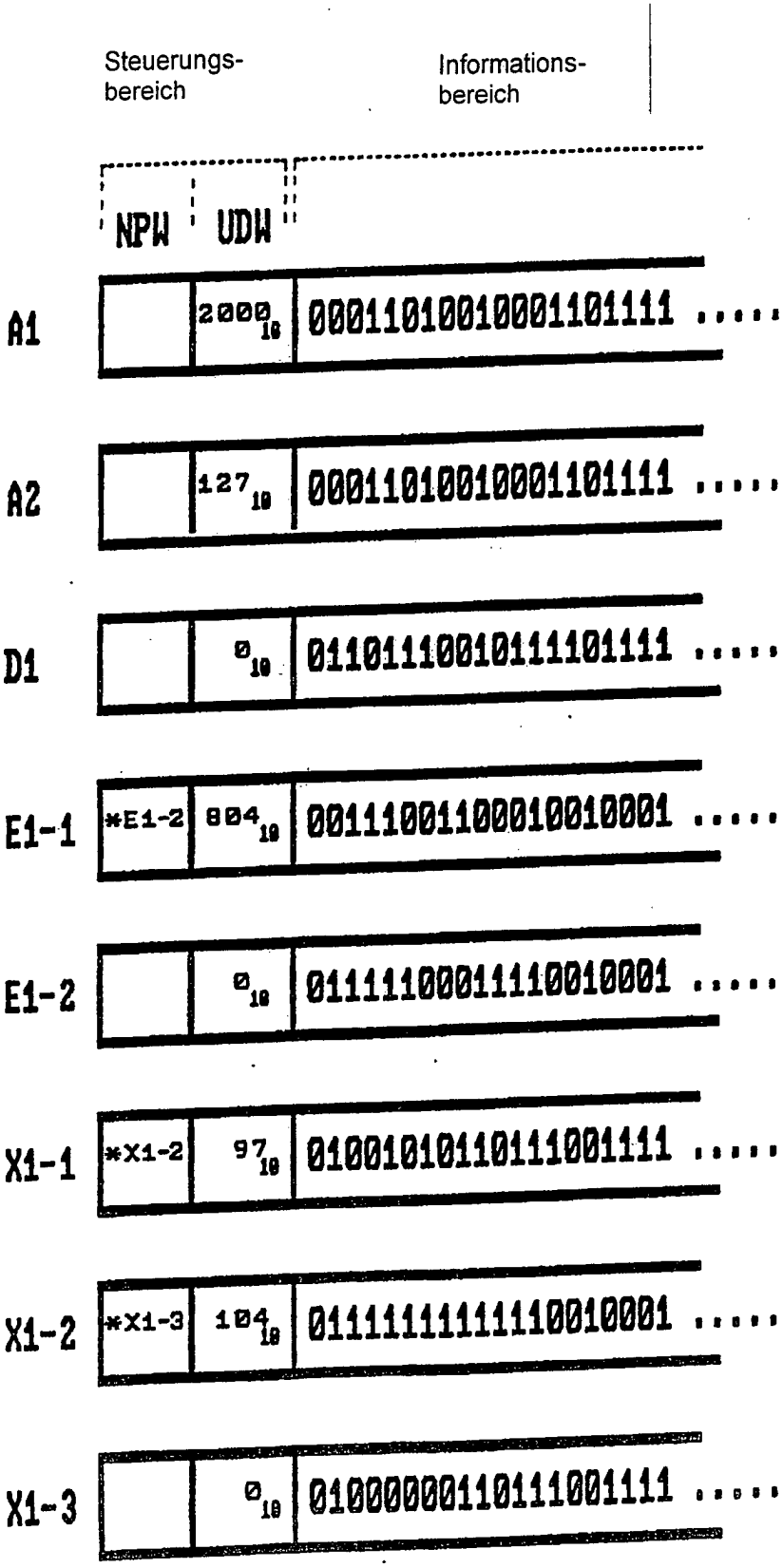


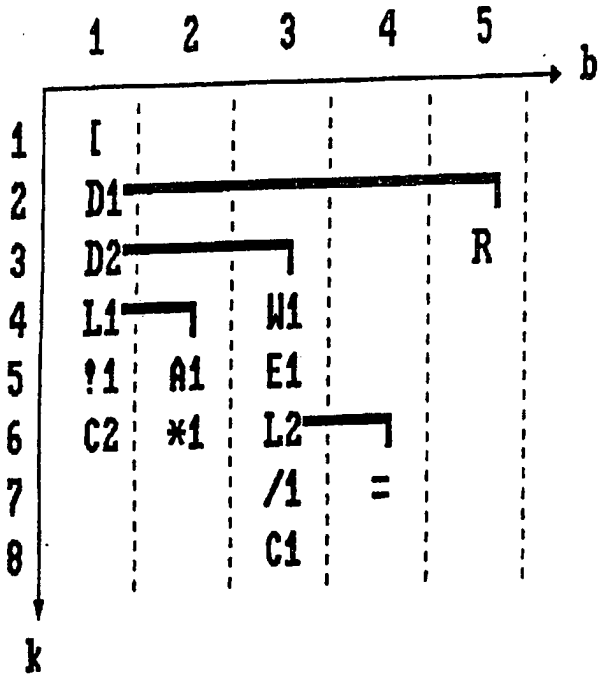
FIGUR 18 B


2
System-ergebnis-geschichte
1
System-eingabe geschichte
3
System-ausführ-rungs-geschichte
19
Eingabe Monitor Puffer
15
vorheriges Bild-schirm-bild
16
14
9
eX-Analyse-maschine
11
eX-Schnittstelle
Inkrementelles Bild Extruktor/ Konstruktor
12
6
7
4
13
Steue-des eX
rungskomponente Objektcodes
20
23
8
15
24
eX-Modell
eX-Treiber
Interpretiere eX-Ver-wahrungsort Anweisung
SYSTEM I/O
5
10
eX-Verwahr-ort
Datei Ausgabe

FIGUR 18 C

FIGUR 19


Steuerungs-
bereich
Informations-
bereich
NPW
UDW
A1
2000 10
0001101001000110 1111 .....
A2
127 10
0001101001000110 1111 .....
D1
0 10
0110111001011110 1111 .....
E1-1
*E1-2
804 10
0011100110001001 0001 .....
E1-2
0 10
0111110001111001 0001 .....
X1-1
*X1-2
97 10
0100101011011100 1111 .....
X1-2
*X1-3
104 10
0111111111111001 0001 .....
X1-3
0 10
0100000011011100 1111 .....



FIGUR 20 A

```
      INTEGER IN(3)
      CHARACTER IA(3), IALF(10), CH
      DATA IALF/'0','1','2','3','4','5','6','7','8','9'/
881   FORMAT(3X,'IN(I)=',I3,' DOESNOT MATCH INTEGER FROM 0 TO 9 ',
     C ' TYPE "Y" TO CONTINUE')
882   FORMAT(A)
D1:   I=1,3
D2:   J=1,10
L1:   IF(IN(I).EQ.J-1)
A1:   IA(I)=IALF(J)
W1:   WRITE(*,881)IN(I)
E1:   READ(*,882)CH
L2:   IF(CH.NE.'Y'.AND.CH.NE.'y')
```

FIGUR 20 B

FIGUR 21 A

```
      SUBROUTINE 12A(IN,IA)
      INTEGER IN(3)
      CHARACTER IA(3),IALF(10)
      DATA IALF/'0','1','2','3','4','5','6','7','8','9'/
      DO 10 I=1,3
        DO 55 J=1,10
          IF(IN(I).EQ.J-1)THEN
            IA(I)=IALF(J)
            GO TO 10
          ENDIF
55      CONTINUE
        WRITE(*,100)IN(I)
100     FORMAT(3X,'IN(I)=',I3,' DOES NOT MATCH INTEGER ',
     C   'FROM 0 TO 9   TYPE "Y" TO CONTINUE')
        READ(*,20)CH
20      FORMAT(A)
        IF(CH.NE.'Y'.AND.CH.NE.'y') STOP
10    CONTINUE
      RETURN
      END
```

FIGUR 21 B

```
      SUBROUTINE 12AXXX(IN,IA)
      INTEGER IN(3)
      CHARACTER IA(3),IALF(10)
      DATA IALF/'0','1','2','3','4','5','6','7','8','9'/
881   FORMAT(3X,'IN(I)=',I3,' DOES NOT MATCH INTEGER ',
     C  'FROM 0 TO 9   TYPE "Y" TO CONTINUE')
882   FORMAT(A)
      DO 3881 I=1,3
        DO 3882 J=1,10
          IF(IN(I).EQ.J-1)THEN
            IA(I)=IALF(J)
      0       CALL eX I(14,2)
            GO TO 23881
          ENDIF
      0   CALL eX I(6,2)
3882  CONTINUE
      WRITE(*,881)IN(I)
      0   CALL eX I(28,2)
      0   CALL eX I(21,2)
      READ(*,882)CH
      0   WRITE(M,882)CH
      IF(CH.NE.'Y'.AND.CH.NE.'y') THEN
      0     CALL eX I(31,2)
        STOP
      ENDIF
23881B    CALL eX I(24,2)
3881  CONTINUE
      CALL eX I(35,2)
      RETURN
      END
```