Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 80 (1989)

Heft: 3

Artikel: ART : Automated Reasoning Tool

Autor: Staab, R.

DOI: https://doi.org/10.5169/seals-903633

ART – Automated Reasoning Tool

R. Staab

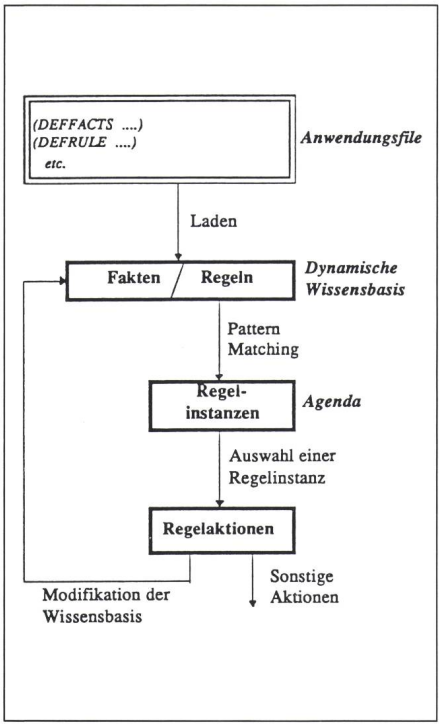
Eines der neben KEE und Knowledge Craft mächtigsten Software-Werkzeuge zur Realisierung wissensbasierter Systeme – ART – wird in diesem Beitrag beschrieben und durch anschauliche Beispiele illustriert. Stärken und Schwächen in der Benutzung werden kritisch beleuchtet, und es werden Vergleiche zu den oben genannten Konkurrenzprodukten gezogen.

Cet article a pour but de décrire l'environnement de développement de systèmes à base de connaissances ART, l'un des plus puissants avec KEE et Knowledge Craft en l'illustrant d'examples explicatifs. Ses points forts et défauts seront ensuite discutés et l'on comparera ART à ses concurrents déjà mentionnés.

Adresse des Autors

Richard Staab, Dipl.-Informatiker, Insiders Gesellschaft für angewandte Künstliche Intelligenz (BDU), D-6500 Mainz.

ART ist ein Produkt der *Inference Corporation*. Die folgende Beschreibung basiert auf der Version 3.0, welche seit 1986 verfügbar ist. ART ist beeinflusst durch die klassischen Systeme *Hearsay-III* [1] und *OPS5* [2]. Obgleich ART auch eine Entwicklungsumgebung umfasst, muss es in erster Linie als eine äusserst facettenreiche *Sprache* verstanden werden, genauer, als ein Konglomerat von mehreren Wissensrepräsentationssprachen. Dazu gehören etwa Sprachen zur Beschreibung von Regeln, von semantischen Netzen, von Frame-Hierarchien und von hypothetischen Welten. Obwohl ART sehr homogen strukturiert ist, führt die Fülle der sprachlichen Mittel zu einer hohen Komplexität. Wie auch bei den anderen hier besprochenen Werkzeugen ist die sprachliche Basis Lisp.



Figur 1 Reasoning-Zyklus von ART

Fakten und Regeln

Wie aus der Figur 1 ersichtlich, wird die dynamische Wissensbasis durch Fakten und Regeln (Tab. I) gebildet. Die Fakten, die in ihrer Gesamtheit den «Weltausschnitt» der Anwendung darstellen, werden in der Form

(RELATION-NAME VALUE 1 VALUE 2...) (1)

gespeichert. Im Ausdruck

(stellung weisse-dame D 4) (2)

zum Beispiel ist *Stellung* ein Relationsname (Relation Name) und *weisse-dame*, D und 4 sind Werte (Values). Die Inferenzmaschine ermittelt die auszuführenden Regeln über einen Pattern-Matching-Prozess. Als Beispiel diene die Regel

```
(DEFRULE drucke-                          (3)
gefährdete-figuren
  (stellung ?figur ?x ?y)
  (bedroht ?figur)
  =>
  (printout t t ''Figur '' ?figur ''
  in Stellung '' ?x ?y))[1]
```

Das Schlüsselwort *DEFRULE*, gefolgt vom Regelnamen, leitet die Definition der Regel ein. Es handelt sich um eine Vorwärtsregel mit der Bedeutung: Wenn *?figur* sich in Stellung *?x ?y* befindet und bedroht ist, dann drukke entsprechende Meldung aus. *(stellung...)* und *(bedroht...)* sind hier Pat-

[1] *Die Fragezeichen kennzeichnen Variablen, und DEF* in *DEFRULE* weist auf «Definition» hin.

terns, die auf mögliche Matches mit Fakten der dynamischen Wissensbasis hin überprüft werden. In unserem Beispiel passt das erste Pattern auf obiges Faktum, wobei die Platzhalter *?figur, ?x* und *?y* an *weisse-dame, D* und *4* gebunden werden. Danach muss ein Faktum *(bedroht weisse-dame)* zum Match gefunden werden, und die Regel kann mit dieser Platzhalterbelegung in einer Agenda zum Feuern bereitgestellt werden. Wenn mehrere Matches mit unterschiedlichen Belegungen möglich sind, werden mehrere Instanzen der Regel bereitgestellt. Nach der prioritätsgesteuerten Auswahl einer Regelinstanz kommt deren Aktionsteil schliesslich zur Ausführung. Hier können Objekte der Wissensbasis erzeugt, modifiziert und gelöscht oder beliebige sonstige ART- bzw. Lispfunktionen gerufen werden. Im Beispiel wird die ART-eigene Druckroutine *printout* aufgerufen (die ersten beiden Argumente *t t* beziehen sich auf das Ausgabemedium und das Druckformat).

Der gesamte beschriebene Vorgang vom Pattern Matching bis zum Feuern wiederholt sich, bis die Agenda irgendwann leer ist. Die Beispielregel druckt auf diese Weise der Reihe nach alle bedrohten Figuren aus. Es bleibt noch anzumerken, dass eine Regel natürlich nicht zweimal auf der Basis der gleichen Faktenkombination feuern kann, sonst würde die Agenda nie leer werden und das System niemals stoppen. Das obige Beispiel offenbart nur einen winzigen Ausschnitt der Möglichkeiten der mächtigen Pattern-Matching-Sprache von ART, die auch Existenzquantoren und Iterationen umfasst. Wie gezeigt wurde, ergibt sich die Dynamik des Systems in erster Linie durch iteratives Feuern von Vorwärtsregeln (Forward Chaining).

In diesen Prozess können Phasen von Backward Chaining eingebettet sein. Dazu werden Rückwärtsregeln definiert, die bestimmte Fakten ableiten, wenn Vorwärtsregeln sie zur Verifizierung ihres Bedingungteils brauchen. ART führt für das Backward Chaining übrigens keinen eigenen Inferenzmechanismus ein, sondern «simuliert» den Vorgang durch Vorwärtsregeln hoher Priorität, angestossen durch besondere Fakten (Goal Facts).

Das Pattern Matching erfolgt in ART mittels einer weiter beschleunigten Version des an sich schon schnellen Rete-Algorithmus. Dies ist einer der Gründe für die bekannten Performance-Qualitäten von ART, bewirkt aber umgekehrt erhöhten Zeitaufwand beim initialen Laden der Wissensbasis, weil der Algorithmus auf einer Kompilierung der Regelbedingungen in Netzwerke beruht. Die Version 3.0 erlaubt es, Regelfiles zunächst in Binärfiles zu kompilieren, welche dann um ein Vielfaches schneller geladen werden können.

Glossar

Agenda:	Liste von feuerbereiten Regelinstanzen
Browsing:	Blättern in einer Wissensbasis
Demon:	im Hintergrund ablaufende Operation, die bei bestimmten Zuständen der Wissensbasis aktiv wird
Faktum:	atomare Wissenseinheit, z.B. (Farbe Auto Blau)
Fokussierung:	Beschränkung der Regelanwendung auf einen Teil der Regelbasis
Frame:	Bündelung von Fakten zu einer semantischen Einheit (vergleichbar mit einem Record in Pascal)
Message:	Nachricht an ein Objekt (objektorientierte Programmierung)
Methode:	zu einem Objekt gehörende Kodeeinheit, welche die Reaktion auf eine Message definiert (objektorientierte Programmierung)
Objektorientierte Programmierung:	Programmierform, bei der nachrichtenaustauschende Objekte im Mittelpunkt stehen
Pattern Matching:	die Abgleichung der Regelbedingungen auf die sich in der Wissensbasis befindenden Fakten
Reasoning:	Schlussfolgerungsprozess über einer Wissensbasis
Regelinstanz:	Ausprägung einer Regel, bei der nach erfolgreichem Pattern Matching die variablen Teile an die entsprechenden Fakten gebunden wurden
Schema:	ART-typisches Konstrukt zur Darstellung von Frames
Taxonomie:	hierarchische Gliederungsform für Wissensbereiche mit baumartiger Begriffsstruktur (z.B. Fauna)
Uncertainties:	Unsicherheiten, mit denen Wissenseinheiten behaftet sein können (z.B. «Daumenregeln»)

Tabelle I

Schemata und Relationen

Die meisten Anwendungen erfordern eine Bündelung der Fakten zu semantischen Einheiten, die häufig auch als Frames bezeichnet werden. In ART werden diese Strukturen als Schemata (Tab. I) bezeichnet. Das folgende Beispiel beschreibt ein UND-Gatter.

(DEFSCHEMA and-gate (4)
(is-a gate)
(number-of-inputs 2)
(number-of-outputs 1)
(function logical-and))

And-gate ist der Name des hier beschriebenen Objektes, während die einzelnen Slots die Eigenschaften beschreiben. Solche Schemata werden von ART in gewöhnliche Fakten übersetzt und in die Wissensbasis eingetragen, im Beispiel in die Fakten

(SCHEMA and-gate is-a gate), (5)
(SCHEMA and-gate
(number-of-inputs 2) usw.

Auf diese Weise wird die frameorientierte Wissensrepräsentation konzeptionell voll in den Pattern-Matching-Mechanismus integriert. Die Relationen in semantischen Netzwerken werden in ART durch die Slots repräsentiert. Ein bestimmtes UND-Gatter *a2* einer Schaltung könnte etwa folgendermassen definiert sein:

(DEFSCHEMA a2 (6)
(instance-of and-gate)
(connected-with a1 o1 n2))

Der Slot *instance-of* etabliert eine Beziehung zu dem oben definierten Schema *and-gate*. In ART gibt es 9 vordefinierte Relationen. Zu ihnen gehören etwa *is-a, instance-of, element-of, subset-of* sowie die jeweiligen inversen Relationen. Daneben kann der Entwickler beliebige eigene Relationen definieren, im Beispiel etwa *connected-with*. Dies erfolgt wieder durch die Definition eines Schemas, denn ART beschreibt die Eigenschaften von Slots und Relationen selbst wieder in Schemata. *Connected-with* könnte folgendermassen definiert werden:

(DEFSCHEMA connected-with (7)
(instance-of relation)
(inverse connected-with))

Die Definition einer inversen Relation (hier ebenfalls *connected-with*) be-

wirkt den automatischen Eintrag des gegenläufigen Slots beim verbundenen Objekt. *a1*, *o1* und *n2* würden im Beispiel durch ART den Eintrag *(connected-with a2)* erhalten. Daneben gibt es die Möglichkeit, Transitivität bei Relationen zu verankern.

Die Vererbung von Objekteigenschaften innerhalb von Begriffshierarchien (Taxonomien) erfolgt in ART entlang besonderer INH-Relationen (INH = inheritance). Die vordefinierte Relation *is-a* ist eine INH-Relation. Daher würden im Beispiel Eigenschaften von *gate* automatisch auf die Ausprägung *and-gate* übertragen werden. Der Entwickler kann das Vererbungsverhalten bei vordefinierten und selbstdefinierten Relationen steuern, insbesondere was den Konfliktfall mehrerer unterschiedlicher vererbter Werte (bei mehreren Oberbegriffen) angeht.

Es sei in diesem Zusammenhang noch auf einen subtilen Unterschied zu KEE hingewiesen. ART unterscheidet nicht zwischen Klassen und Instanzen. Beides sind in ART Schemata. Semantische Unterschiede ergeben sich nur über die Eigenschaften der Relationen *is-a* (Unterklasse) und *instance-of* (Instanz), auch dies ein Beispiel für die Bestrebung nach konzeptioneller Geschlossenheit.

Zur Darstellung (Browsing) semantischer Netze bietet ART eine graphische Oberfläche an.

Objektorientierte Programmierung und Active Values

Seit Version 3.0 erlaubt ART die objektorientierte Programmierung. Objekte sind die besprochenen Schemata (Frames). Die Methoden werden durch ein Konstrukt *DEFACTION* in Lisp-Syntax definiert. Messages erfolgen syntaktisch wie Aufrufe von Lispfunktionen und integrieren sich gut in die Regelsprache. Eine Erweiterung des bekannten Flavor-Konzeptes sind Multi Methods, die für eine beliebige Kombination von Objekten definiert werden können.

Unter Active Values versteht man den automatischen Anstoss von Aktionen bei bestimmten Operationen auf Slots, sie werden in ART objektorientiert realisiert. Es stehen für acht Arten des Slotzugriffs Varianten bereit. Beliebig viele Active Values können an einen Slot gebunden werden, und sie werden mit ihm vererbt.

Viewpoints

In vielen Applikationen ist eine Teilung der Wissensbasis in verschiedene hypothetische Welten sinnvoll. Hierfür bietet ART den Viewpoint-Mechanismus an. Die Wissensbasis wird dabei in Form eines Netzwerkes von Kontexten gesehen. In jedem Kontext gelten bestimmte Fakten. Beim Übergang in eine neue Hypothese bleiben im allgemeinen die meisten Fakten gültig, so dass sie vom Vaterkontext geerbt werden können (Effizienz). Die Gültigkeit eines Faktums in einem bestimmten Kontext wird in ART durch eine raffinierte Indizierung des Faktums repräsentiert, so dass es nach wie vor nur *eine* Wissensbasis gibt. Regeln können in ihren Aktionen auf bestimmte Kontexte fokussiert werden, aber auch global agieren (Metaregeln). Jedoch verlangen sowohl die Navigation im Netzwerk als auch das subtile Vererbungsverhalten (Kontexte können auch wieder verschmelzen) einen erfahrenen Entwickler, auch wenn die graphische Aufbereitung des Netzwerkes eine gute Unterstützung bietet. Mit dem Konstrukt *DEFCONTRADICTION* können Demons definiert werden, die logisch inkonsistente Kontexte verbieten.

Eine andere typische Anwendung ist das Reasoning in Time (z.B. in Planungsproblemen). Hier repräsentiert ein Kontext einen Weltausschnitt zu einem bestimmten Zeitpunkt. Der Vererbungsmechanismus hilft hier bei der Lösung des bekannten Frameproblems [3]. Viewpoints können auch in mehreren Ebenen benutzt werden, wenn etwa in verschiedenen Hypothesen eines Diagnoseprozesses zeitliche Abläufe simuliert werden sollen. ART unterstützt dies in beliebiger hierarchischer Tiefe, obgleich die Anwendung von mehr als zwei Ebenen schwer vorstellbar wird.

Truth Maintenance

Beim Nonmonotonic Reasoning ist zur Wahrung einer konsistenten Wissensbasis nach Löschen eines Faktums *X* oft eine Löschung aller von *X* abgeleiteten Fakten erwünscht. Dies erfordert eine Buchhaltung des Systems über die gegenseitigen logischen Abhängigkeiten der Fakten [4]. ART erlaubt die Kennzeichnung beliebiger Bedingungen einer Regel als *LOGICAL*, wodurch Abhängigkeiten zwischen triggernden und erzeugten Fakten der Regel verbucht werden. Bei späterem Löschen eines der triggernden Fakten leitet ART dann automatisch die Löschung aller abgeleiteten Fakten ein. Dieser Mechanismus bezieht sich nur auf *erzeugte* Fakten, d.h., durch eine Regel gelöschte Fakten werden nicht erneut instanziert, wenn die triggernden Fakten ungültig werden.

Fehlende Wissensrepräsentationsformen

ART bietet keine direkte Unterstützung für Reasoning mit Uncertainties (unsicherem Wissen) an. Es liegen zwar Beispiele vor, wie bekannte Konzepte (Bayesian Probabilities, Fuzzy Set Theory, Dempster-Shafer, Emycin Model) in ART angewandt werden können [5], jedoch erfordert das einen versierten Entwickler.

Weiterhin steht kein direktes Mittel zur Fokussierung auf gewisse Teile der Regelbasis bereit, was aber leicht über Steuerfakten emuliert werden kann.

Graphische Fähigkeiten

Basis der Graphik in ART sind Icons, die ebenfalls als Schemata definiert sind. ART beinhaltet eine beliebig erweiterbare Graphikbibiliothek. Der Gebrauch, insbesondere die Ansteuerung von Graphiken aus den Regeln heraus, basiert auf dem oben beschriebenen Schema-Mechanismus. Zur schnellen Kreation einer graphischen Oberfläche (inklusive Bitmap-Graphiken) dient der Icon Editor ARTIST. Der Entwickler hat über das Konstrukt *DEFICON* die Möglichkeit, die Graphikbibliothek (und damit auch ARTIST) um selbstdefinierte Icons zu erweitern, jedoch muss er sich dazu in Lispgefilde begeben. Die Animation der Graphiken kann mit Version 3.0 bequem über Active Values erfolgen.

Insgesamt sind die Graphikmöglichkeiten in ART bescheidener als in KEE, genügen jedoch im allgemeinen zur Schaffung geeigneter Benutzeroberflächen.

Schnittstellen

Schnittstellen zu anderen Programmen und Systemen müssen vom Entwickler selbst in Lisp programmiert werden. Dabei hilft die gute Einbettung von Lisp in die Regelsyntax, aber auch eine funktional reich ausgestattete Lisp-Oberfläche von ART. Alle wesentlichen Operationen in ART (z.B. RESET der Wissensbasis) sind von

Lisp aus möglich, insbesondere auch Operationen auf der Wissensbasis selbst. So können in Realzeitanwendungen etwa Lisp-Programme dem simultan laufenden ART asynchron Fakten zuspielen, die dort unmittelbar das Systemverhalten beeinflussen.

Dennoch wären Schnittstellen sowohl zu prozeduralen Sprachen als auch zu Datenbanksystemen wünschenswert.

Die Entwicklungsoberfläche

Die Entwicklungsoberfläche von ART ist für kleine bis mittelgrosse Anwendungen funktional und ergonomisch voll ausreichend. Hervorzuheben sind hier Dinge wie inkrementelle Kompilation, gute Browsing- und Trace-Möglichkeiten und Hilfe bei der Leistungsverbesserung von Anwendungen.

Wird aber die Wissensbank im Laufe einer Entwicklung wirklich gross, werden bekannte Probleme des Software-Engineerings spürbar, gerade bei einem Werkzeug, welches als Programmiersprache verstanden werden muss. Der Aufbau einer Applikation erfolgt derzeit zum grössten Teil durch Eintippen der Definitionen im ZMACS-Editor. Effizienter wäre die Eingabe über vorgegebene Masken (etwa bei Schemata und Regeln). Für Knowledge-Engineering-Zwecke sollte unbedingt auch Platz für deskriptive Nebeninformation zur Verfügung stehen (z.B. Quellenangaben). Auch der schnelle Zugriff auf zu editierende Objekte ist zu verbessern. Konsistenzprüfungen über der Wissensbasis werden nicht direkt unterstützt. In [6] wird allerdings eine umfassende Konsistenzprüfungskomponente speziell für ART beschrieben.

Kein Garbage

Ein besonderes Problem von Werkzeugen auf Lispmaschinen ist das nicht vorhersehbare Laufzeitverhalten. Dies liegt daran, dass durch laufende Programme im Speicher i.a. fortwährend Garbage erzeugt wird, der durch einen speziellen Hintergrundprozess (Garbage Collection) simultan wieder abgebaut werden muss. Da dieser Prozess der eigentlichen Applikation Rechenzeit entzieht, ist es schwierig, Reaktionszeiten zu garantieren. Wenn darüber hinaus mehr Garbage produziert als abgeräumt wird, kann die Applikation durch «Seitenflattern» im virtuellen Speicher praktisch zum Stillstand kommen. Dies war nach unseren Erfahrungen auch bei ART 2.0 möglich. Es ist daher von grösster Bedeutung, dass ART ab Version 3.0 durch eine eigene Verwaltung der Speicherressourcen praktisch garbagefrei ist.

Zusammenfassung

Den Entwicklern von ART ist es erstaunlich gut gelungen, die unterschiedlichsten Methoden, die die KI-Forschung in den letzten zehn Jahren hervorgebracht hat, auf eine konzeptionelle Schiene zu bringen. ART ist sehr homogen, kein Feature der Sprache wirkt improvisiert oder unausgereift. Auch Dokumentation und Schulungsmaterialien sind solide. Eine wirkliche Nutzung der Möglichkeiten von ART setzt aber einen erfahrenen Knowledge Engineer mit guten Lispkenntnissen voraus. Auch um das nicht immer einfache Verhalten der Inferenzmaschine zu verstehen, sollte er fundierte Kenntnisse auf dem Gebiet der KI besitzen. Für viele Anwendungen (Realzeit) kann die unerreichte Schnelligkeit in Verbindung mit der Garbagefreiheit von grosser Bedeutung sein. Bei grossen Wissensbasen wird dem Entwickler eine disziplinierte Arbeitsweise abverlangt, insbesondere was die Strukturierung und die Dokumentation angeht.

Zur Unterstützung bei der Einbindung in die konventionelle Datenverarbeitung muss noch einiges getan werden. Auch bezüglich der Portierung auf andere Hardwareplattformen hinkt ART seinem Konkurrenten KEE hinterher.

Zusammenfassend kann ART als schnelles, elegantes aber auch komplexes Werkzeug bezeichnet werden, besonders geeignet zur Lösung schwieriger Problemstellungen, bei denen eine ausgeprägte Dynamik mit hohen Anforderungen an das Regelverhalten und nicht so sehr die statische Repräsentation grosser Wissensbereiche im Vordergrund steht.

Literatur

[1] *C.L. Forgy:* The OPS5 user's manual. Technical Report CMU-CS-81-135. Pittsburgh/Pa, Carnegie-Mellon University, Computer Science Department, 1981.

[2] *L.D. Erman, P.E. London* and *S.F. Fickas:* The design and an example use of Hearsay-III. Proceedings of the seventh International Joint Conference on Artificial Intelligence (IJAI-81) 24...28 August 1981, Vancouver, Canada. Vol. 1, p. 409...415.

[3] *E. Charniak* and *D. McDermott:* Introduction to artificial intelligence. Reading/Mass., Addison-Wesley, 1985.

[4] *J. Doyle* and *P. London:* A selected descriptor-indexed bibliography of the literature on belief revision. Sigart Newsletter -(1980)71, p. 7...23.

[5] *M.McFall:* Representing uncertainties in ART. Los Angeles/California, Inference Corporation.

[6] *T.A. Nguyen:* ARC: The ART rule checker. Palo Alto, Lockheed Research & Development, Advanced Software Laboratory.