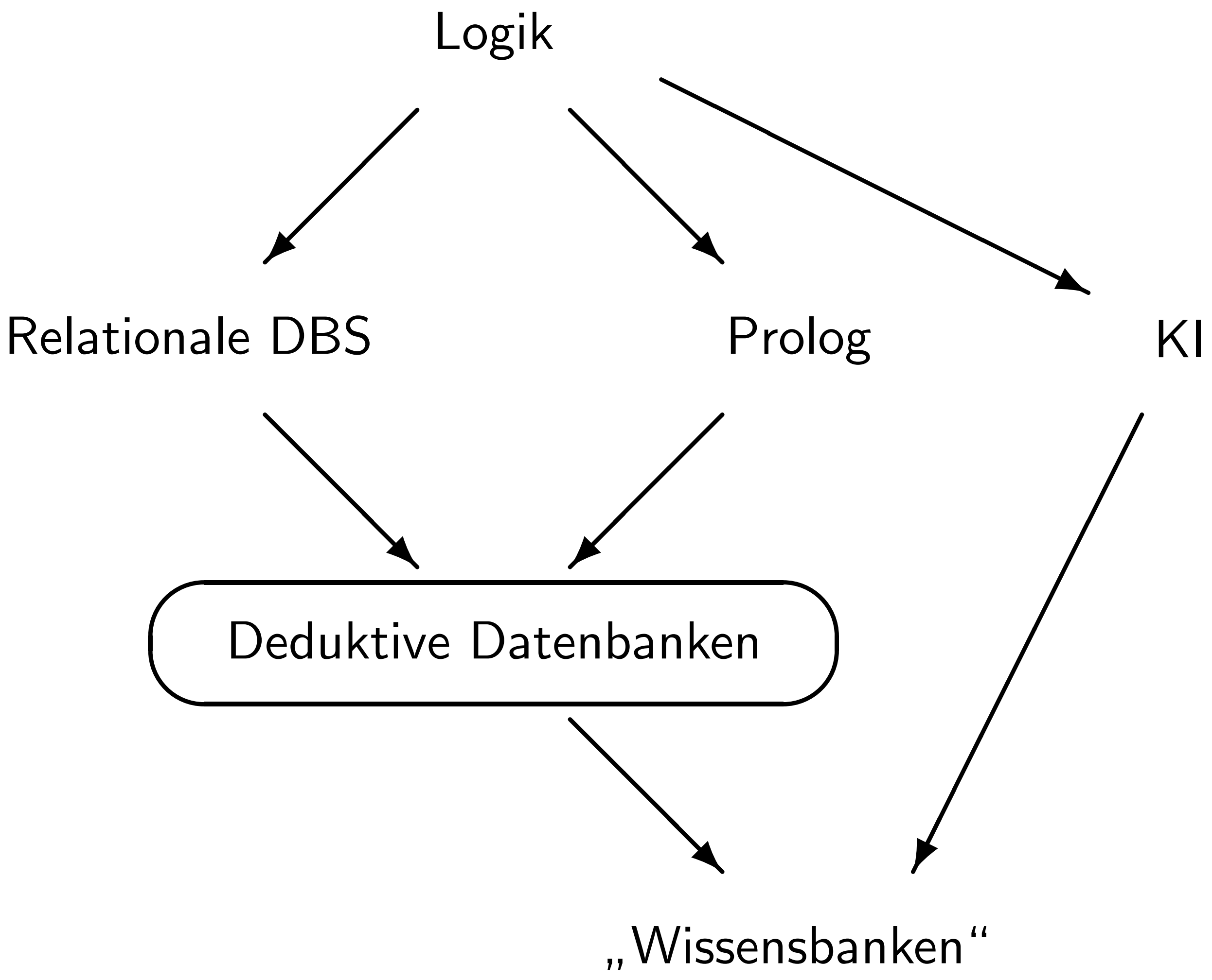
1. Einleitung

1. Einordnung des Gebietes, Geschichtliche Entwicklung
2. Beispiel
3. Definitionen/Sichtweisen deduktiver Datenbanken
4. Motivation für integrierte Systeme aus PS + DB
5. Deklarative Programmierung
6. Stärken deduktiver Datenbanken ($\rightarrow$ Anwendungsbereiche)
7. Deduktive Datenbanken in der Krise?

Einordnung des Gebietes



Deduktive DBen (1)

Verschiedene Sichtweisen:

- Integriertes System aus relationaler DB und Prolog-ähnlicher Programmiersprache.
- „Pure Prolog“ mit besonderer Unterstützung zur Verwaltung großer Faktenmengen.
- Relationale Datenbank mit
  neuer Anfragesprache (Datalog statt SQL) und
  der Möglichkeit, Sichten („views“) rekursiv zu definieren.
- Eingeschränkter automatischer Beweiser:
  Nur sehr einfache Formeln, aber davon sehr viele
  („deduzieren“ = logisch folgern).

Deduktive DBen (2)

Eine deduktive Datenbank besteht aus:

- einer relationalen Datenbank,
  die Relationen/Prädikate „extensional“ definiert,
  d.h. durch Aufzählung der Tupel in den Relationen, und
- einem logischen Programm,
  das Relationen/Prädikate „intensional“ definiert,
  d.h. durch Angabe von definierenden Regeln
  (logischen Formeln einer bestimmten Art).

Beispiel für extensional $\leftrightarrow$ intensional: Fahrplan.

Geschichtliche Entwicklung

1963	Resolventen-Methode (Robinson)
∼1969	Question Answering Systems (Green et.al.)
1970	Relationales Datenmodell (Codd)
1973	Prolog (Colmerauer, Roussel, et.al.)
1976	Minimales Modell (van Emden, Kowalski)
1977	Konferenz „Logic and Databases“ (Minker et.al.)
1982	Beginn des „Fifth Generation Projects“ in Japan
1986	„Magische Mengen“, Perfektes Modell, Erste Systeme
∼1989	Erste Lehrbücher

Beispiel (1)

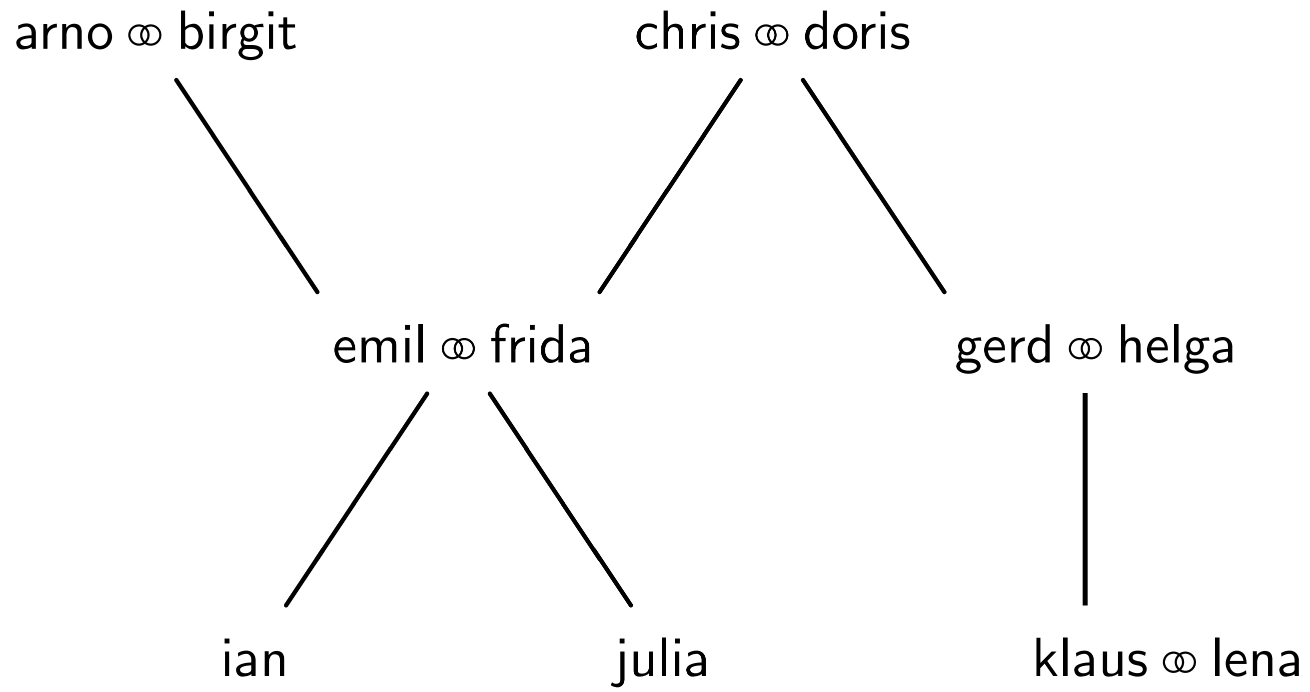
Datenbank:

eltern		
Kind	Vater	Mutter
emil	arno	birgit
frida	chris	doris
gerd	chris	doris
ian	emil	frida
julia	emil	frida
klaus	gerd	helga

ehepaar	
Mann	Frau
arno	birgit
chris	doris
emil	frida
gerd	helga
klaus	lena

Beispiel (2)

DB (graphisch):



Beispiel (3)

DB (als Prolog-Fakten):

```
% Kommentar: eltern(Kind, Vater, Mutter).
eltern(emil, arno, birgit).
    ⋮
eltern(klaus, gerd, helga).

% ehepaar(X,Y)   ⟺   X und Y sind verheiratet,
%                    X ist Mann, Y ist Frau.
ehepaar(arno, birgit).
    ⋮
ehepaar(klaus, lena).
```

Beispiel (4)

Programm:

$$\begin{array}{lcl}
\text{elternteil}(X, Y) & \leftarrow & \text{eltern}(X, Y, _). \\
\text{elternteil}(X, Y) & \leftarrow & \text{eltern}(X, _, Y). \\
\text{großelternteil}(X, Z) & \leftarrow & \text{elternteil}(X, Y) \wedge \\
 & & \text{elternteil}(Y, Z).
\end{array}$$

Anfragen:

? großelternteil(julia, X).

? großelternteil(X, arno).

? großelternteil(klaus, arno).

? großelternteil(X, Y).

Beispiel (5)

Aufgaben:

- Definieren Sie das Prädikat „verheiratet“ (symmetrisch: Wenn X mit Y verheiratet ist, so auch Y mit X.)
- Definieren Sie das Prädikat „geschwister“ (dabei kann als Voraussetzung auch $X \neq Y$ verwendet werden).
- Definieren Sie das Prädikat „onkel_tante“.
- Definieren Sie das Prädikat „vorfahr“.
- Definieren Sie das Prädikat „mein_vorfahr“ (für julia).
- Definieren Sie ein Prädikat zur Konsistenzprüfung (Ist eine Person gleichzeitig männlich und weiblich?).

Datalog vs. Prolog (1)

- Weniger Kontrolle, mehr Logik:
  - Kein Cut
  - Reihenfolge der Regeln irrelevant
  - Reihenfolge der Rumpfliterale nicht unbedingt relevant
    Je nach System und eingestellten Optimierungen.
- Grund:
  - Prolog-Programmierer kennt Verwendung der Prädikate.
    Es gibt nur ein Haupt-Prädikat, das alle anderen aufruft.
  - Datenbanken erlauben sehr unterschiedliche Anfragen.
    Optimierer muß je nach Ein-/Ausgabe-Argumenten die beste Reihenfolge der Rumpfliterale wählen.

Datalog vs. Prolog (2)

- Besseres Terminierungsverhalten:
  - Terminierung garantiert, falls keine Listen, Funktionssymbole und eingebauten Prädikate.
  - Unterschied DB-Sprache vs. Programmiersprache: Garantierte Terminierung.
  - Heutige DDB-Systeme erlauben i.w. volles Pure Prolog.
- Datalog-Implementierungen besser für große Faktenmengen.

  Wenn die Fakten auf dem Externspeicher liegen, bietet sich eine mengenorientierte Auswertung an, da man immer einen ganzen Block lesen muß.

Datenbank + Programmiersprache (1)

Vorteile/Funktionen von Datenbanken:

- Persistenz
  (Daten leben länger als Programmausführung/Prozeß).
- Integration, Vermeidung von Redundanz
- Datenunabhängigkeit
  (Lastprofil ändert sich, Daten leben länger als Programme!)
- Spart Programmieraufwand: Viele Algorithmen eingebaut, besonders für Externspeicherverwaltung (Suchen/Sortieren).
- Ad-hoc Anfragen, auch für ungeübte Benutzer

Datenbank + Programmiersprache (2)

Vorteile/Funktionen von Datenbanken, Forts.:

- Große Datensicherheit (Backup&Recovery)
- Zusammenfassung von Änderungen zu Transaktionen (ganz oder gar nicht)
- Mehrbenutzerbetrieb (Synchronisation gleichzeitiger Zugriffe)
- Integritätssicherung
- Benutzersichten
- Datenschutz/Zugriffskontrolle
- Systemkatalog (Data Dictionary), Anfragen an Meta-Daten

Datenbank + Programmiersprache (3)

DB ohne Programmiersprache kaum denkbar:

- Für den Endanwender unzumutbar: Immer gleiche Muster von SQL-Anweisungen (nur mit unterschiedlichen Konstanten).
- Nicht berechnungsuniversell, z.B. in SQL nicht möglich:
  - transitive Hülle, „Nicht-Standard-Aggregationen“
  - komplexere numerische Berechnungen
  - Dateiverarbeitung, Formatumwandlung, Anschluß an fremde Systeme
- DB-Anwendungen brauchen moderne graphische Benutzeroberflächen.

Integrierte Systeme (1)

Allgemeine Tendenz im DB-Bereich:

- Persistente Programmiersprachen: Pascal/R, PJava
- Objekt-orientierte Datenbanken (C++ & Persistenz)
- Prozeduren in der DB ablegbar: Oracle PL/SQL
- Trigger in Datenbanken (aktive Datenbanken)
- Sprachen der vierten Generation (4GL)
- Deduktive Datenbanken
  Besonderheit: Deklarative Programmiersprache

Integrierte Systeme (2)

Probleme bei Embedded SQL:

- Programmierer muß mit zwei (oder mehr) unterschiedlichen Sprachen und Compilern arbeiten
- Unterschiedliches Typsystem in C und SQL
- „Impedence Mismatch“: Mengenorientierte Verarbeitung in SQL, Tupelorientierte Verarbeitung in C
- Optimierer der Datenbank kann nur einzelne Anfragen optimieren (Programm zwischen den Anfragen unbekannt)
- Aufbewahrung von Zugriffsplänen für Anfragen in Programmen notwendig und nicht einfach.

Integrierte Systeme (3)

Integration ist natürliche Entwicklung:

- Bisher wurden mit DBMSen die Daten einer Firma unter eine zentrale Kontrolle gestellt:
  - um Redundanzen und Inkonsistenzen zu vermeiden,
  - die Daten besser gemeinsam zu nützen,
  - und neue Anwendungen leichter zu machen.
- Jetzt auch Programme / Prozeduren.

Integrierte Systeme (4)

Unterprogramm-Bibliothek ist nicht dasselbe:

- Wurde eine Prozedur der Bibliothek geändert, mußte die neue Version explizit zu allen Anwendungsprogrammen hinzugebunden werden.
- Falls man das DB-Schema ändert, werden eventuelle Probleme erste entdeckt, wenn das entsprechende Anwendungsprogramm ausgeführt wird.
- Anwendungsprogramme und die von ihnen benutzen Relationen sind nicht im Data Dictionary vermerkt.
- Solche Prozeduren sind nicht in Anfragen und Triggern verwendbar. Sie existieren nur außerhalb des DBS.

Integrierte Systeme (5)

Vorteile für Client-Server:

- Bei normaler Anwendungsprogrammierung müssen mehr Daten über das Netz geschickt werden, weil die DB auf dem Server und die Anwendung auf dem Client läuft.
- Geschieht die Verarbeitung der Daten dagegen auf dem Server, so müssen nur die Ergebnisse zum Client übertragen werden.
- Der Server-Rechner ist meist wesentlich leistungsfähiger.
- Prozeduren auf dem Server können zentral administriert werden, Anwendungsprogramme auf den Clients müssen auf jedem einzeln geändert werden.

Integrierte Systeme (6)

Vorteile für Konsistenz:

- Wenn die Daten nur über vorgegebene Prozeduren geändert werden können, kann Konsistenz leichter garantiert werden (wie bei abstrakten Datentypen, Objektorientierung).
- Anwendungen von Triggern (Prozeduren im DBMS, die bei Ereignissen wie z.B. Updates angestoßen werden):
  - Integritätsprüfung
  - Aktualisierung abgeleiteter Daten
  - benutzerdefinierte Protokollierung
  - komplexe Zugriffsrecht-Prüfungen
  - Anstoßen von Geschäftsprozessen (z.B. Nachbestellung)

Deklarativität (1)

- Nicht wie (berechnet man die Lösung), sondern was (ist das Problem).
- Algorithmus = Logik + Kontrolle (Kowalski)
  Imperative Programmierung: Kontrolle explizit, Logik implizit.
  Deklarative Programmierung: Logik explizit, Kontrolle implizit.
- Deklarativität ist nichts neues in Datenbanken:
  SQL ist eine deklarative Anfragesprache.
- Häufig einfachere Formulierungen (Benutzer braucht sich über schnelle Auswertung keine Gedanken mehr zu machen) (zehnfach kürzere Formulierungen als in C nicht untypisch)
  $\Rightarrow$ geringere Kosten für Programmerstellung und Wartung.

Deklarativität (2)

- Einfaches und solides theoretisches Fundament:
  Datalog kann auf ca. einer Seite vollständig definiert werden.
- Macht mächtige Optimierer möglich und nötig.
- Programme werden unabhängiger von der aktuellen Hardware/Software Technologie
  - z.B. Parallelisierung einfacher
  - Programme von heute können später die Algorithmen von morgen verwenden
- Eine Logik, viele Algorithmen
- Verifikation einfacher

Stärken deduktiver Datenbanken (1)

Rekursion:

- Transitive Hülle ausdrückbar
- Verarbeitung hierarchisch/Graph-artig strukturierter Daten
- Z.B. Hypertext/WWW ist gerichteter Graph von Dokumenten
- „Object Exchange Model“ etc. auch gerichteter Graph
- Aufrufstruktur von Prozeduren auch gerichteter Graph
- Dokumente sind häufig hierarchisch strukturiert
- Technische Entwurfsgegenstände werden hierarchisch aus immer größeren Baugruppen zusammengesetzt.

Stärken deduktiver Datenbanken (2)

Regeln:

- Experten-Wissen
- Für regulär strukturierte Daten (z.B. Fahrplan)
- Syntax-Regeln
- Integritätsbedingungen
- Aktive Regeln, Trigger
- Wissens-Unabhängigkeit
  Daten-Unabhängigkeit bedeutete, daß Daten unabhängig von Anwendungsprogrammen sind. Nun möchte man aber auch weiteres Wissen über den Anwendungsbereich zentral sammeln.

Stärken deduktiver Datenbanken (3)

Sichten:

- Sichten/abgeleitete Prädikate sind das Basis-Konzept bei deduktiven DBen, bei relationalen DBen nur Zusatz.
- Deduktive DBen erlauben auch Sichten, die nur mit bestimmten Eingabeargumenten aufgerufen werden können (für Berechnungen, Anschluß externer Programme).
- Sichten sind wichtig zur Integration mehrerer heterogener Datenquellen (deduktive DBen als „Middleware“).
- Abspeicherung wiederkehrender Anfragen ($\rightarrow$ Sammlung von Wissen)

Stärken deduktiver Datenbanken (4)

Sichten, Forts.:

- Anpassung des Schemas an die Wünsche verschiedener Benutzer, Datenschutz, logische Datenunabhängigkeit.
- Analyse großer Datenmengen mittels abstrakterer Konzepte (für „Data Mining“, „Decision Support“)

Mehrere Lösungen / Backtracking:

- Muster können in Texten mehrfach auftreten.
- Hypothetisches Schließen: Verfolgen mehrerer Alternativen.

Stärken deduktiver Datenbanken (5)

Komplexe Objekte:

- Man kann u.a. Listen, variante Records, Bäume als Prolog-Terme ausdrücken.
- Derartige Datenstrukturen können in Prolog sehr gut (einfach) verarbeitet werden.

Mustervergleich:

- Man kann auch Terme mit Variablen (Muster für Datenstrukturen) in Prolog verwenden.

Krise deduktiver Datenbanken (1)

- Es gibt sehr viele Forschungsarbeiten.
- Es gibt einige Prototyp-Implementierungen.
  Sie werden aber fast nur im universitären Bereich verwendet (mit wenigen Ausnahmen).
- Es gibt zur Zeit kein kommerziell vermarktetes System.
- Eine Firma mußte schließen, ein (oder mehr?) Projekte wurden gekippt.
- Bei einer Podiums-Diskussion auf der ICDE'93 („Are we polishing a round ball?“) wurde der Bereich „rekursive Anfrageauswertung“ als nicht mehr interessant für innovative Forschung erklärt.

Krise deduktiver Datenbanken (2)

- Ramakrishnan hat 1995 erklärt, daß bessere Implementierungen die aktuellen Probleme lösen würden, und keine fundamental neuen Techniken mehr nötig sind.
- Es gibt kaum noch Forschungsgelder und Professoren-Stellen in diesem Bereich. Immer mehr DDB-Forscher wandern in andere Bereiche ab.
- Prolog-Implementierungen sind immernoch um mehr als eine Größenordnung schneller als deduktive Datenbanken.
- Während Prolog-Compiler typischerweise in Prolog geschrieben werden, wird Datalog überhaupt nicht bei der Implementierung deduktiver DBen eingesetzt.

Gründe, nicht aufzugeben (1)

- Prolog wird in der Industrie erfolgreich eingesetzt (Boeing, British Airways, Kodak, IBM (Machine Translation), Philips Research, SRI (natural language), Swiss Life).
- Bei Ericsson wurde ein spezielles deduktives DBS für den Einsatz im Hause entwickelt.
- Constraint Logic Programming ist in der Industrie auch sehr erfolgreich.
- Viele Ideen von deduktiven Datenbanken fließen jetzt in klassische relationalen DBen ein.
- Viele bekannte DB-Forscher haben auch auf dem Gebiet deduktiver DBen gearbeitet.

Gründe, nicht aufzugeben (2)

- Es ist offensichtlich noch mehr Forschung nötig, damit Datalog wirklich auch als Programmiersprache verwendet werden kann (ohne die Deklarativität aufzugeben).
- Mit Web-Anfragesprachen und intelligenten Agenten im WWW eröffnet sich ein neues Anwendungsfeld, für das Datalog gut geeignet erscheint.
- Man hat inzwischen erkannt, das reine objektorientierte Datenbanken in mancher Hinsicht ein Rückschritt sind. DOOD-Systeme sind vielversprechend.
- Es sind noch mehr in deduktiven Datenbanken ausgebildete Studenten nötig!