Knowledge-Based Systems and Deductive Databases

Wolf-Tilo Balke
Christoph Lofi
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de
6.1 Herbrand Models

6.2 An Operative Semantics

6.3 5th Generation Project

\[
\text{ancestor}(X,Y) \leftarrow \text{parent}(X,Y) . \\
\text{ancestor}(X,Y) \leftarrow \text{ancestor}(X,Z), \text{ancestor}(Z,Y).
\]
• Satisfiability in **Boolean logic is decidable**
  – For any given set of Boolean formulas, there is a algorithm which reliably **tests for satisfiability**
    • This problem is known as **SAT**
    • Still, SAT is **NP-complete**
    • Applicable algorithm: Davis-Putnam
  – The restriction of SAT to **Horn formulae** (i.e. a set of Horn clauses) results in the Horn-SAT problem
    • Horn-SAT is **P-complete**
6.0 Complexity of Logics

• In general satisfiability in first order logics is undecidable
  – There is no algorithm which can check the satisfiability of any first order logic formula in finite time
  – There are semi-decidable algorithms
    • If a given formula is unsatisfiable, the algorithm will find out in finite time
    • If the formula is satisfiable, the algorithm will run forever
  – First order logic can be restricted to a decidable subset
    • E.g., Schönfinkel-Barnays formulas
    • However, complexity is prohibitive for general application in a DB (SB-SAT ∈ NEXP)
6.0 Datalog

• **Datalog** is an implementation of a logical programming language
  – Similar, but less powerful than **Prolog**
  – **Datalog** is restricted to horn clauses
    • **Fact horn clauses** provide the data of the extensional database
    • **Definite horn clauses** provide rules
    • **Goal horn clauses** with only a single literal are used to state queries
• **Datalog** can be further classified
  – Datalog with no functions and negations
  – Datalog\(^\text{neg}\) with negations
  – Datalog\(^f\) with functions
  – Datalog\(^f,\neg\) with both

• It is possible to construct programs where predicate definitions rely on **cyclic negations**
  – This kind of program must be detected and **rejected**
  – To detect this, the program needs to be **stratified**
    • i.e.: do the predicates depend negatively on each other in an **hierarchical** fashion or a **circular** fashion?
6.1 Semantics of Datalog

• What do Datalog programs mean and how can queries be evaluated?
  – To avoid unpleasantness with negation: Datalog

• Remember: a program is given by a set of fact and rule horn clauses
  – A query is given by a goal clause
  – A set of DB-clauses \( \mathcal{W} \) and a query \( Q \equiv L_1, \ldots, L_n \) are unsatisfiable, iff \( \mathcal{W} \models \neg Q \) with \( \neg Q \equiv \exists (L_1 \land \ldots \land L_n) \)
• Thus, for evaluating a program and a query, an operator for \textit{semantic conclusions} is needed
  – We need to find some \textit{interpretation} which is a \textit{model}
  – And it should be \textit{decidable}
  – And it should be \textit{efficient} to find
  – And…
Because Datalog programs consist of clauses, we can use **Herbrand** interpretations.

Remember:

- Herbrand interpretations interpret all constants, functions, and terms **purely syntactical** as themselves.
- The set of all possible atoms of an Herbrand interpretation is called **Herbrand base**.
6.1 Semantics of Datalog

– Different Herbrand interpretations only differ on which elements of the base are true and which are false

– Thus, each Herbrand interpretation can be identified with some subset of the Herbrand base
  • That means a set of atoms with all variables substituted by ground terms
6.1 Semantics of Datalog

• A Herbrand interpretation, thus, **abstracts** from the real world interpretation
  – It purely works on the **symbols** for constants and functions
  – For the **predicate** is has to come up with a set of atoms
    • That are only **consistent atoms** from the Herbrand base
    • That are **all atoms** from the Herbrand base making the formulas of a Datalog program true
  – That means any operator to derive this interpretation has to be **sound and complete**
• Idea:
  – All given facts (which are a subset of the Herbrand base) should all be interpreted as true
    • E.g., $I(Q(x)) = true$
      iff there is a fact rule $Q(x) :- \leftrightarrow (true \rightarrow Q(x))$
  – Furthermore, rules can propagate truth values, iff all premises are true
    • Given a Datalog rule $B :- A_1, A_2, ..., A_n$
      then $B$ has to evaluate to true, if all $A_1, ..., A_n$ are true
6.1 Semantics of Datalog

• This allows us to define semantic conclusion of a datalog program $\mathcal{P} \models W$

  – $W$ is the consequence of the set of Datalog clauses iff each Herbrand interpretation satisfying each clause in $\mathcal{P}$ also satisfies $W$

• or: $W$ is a semantic consequence of $\mathcal{P}$, iff every Herbrand model of $\mathcal{P}$ is also a Herbrand model for $W$
6.1 Example

• **Example** for interpretations and models
  
  – Let’s assume we have only two constants \{Hektor, Christoph\} and two predicates \{green(x), frog(x)\}
  
  – Consider the program:
    
    green(Hektor).
    frog(X) :- green(X).

  – We can come up with several interpretations
    
    • Basically due to the closed world assumption all X in the atoms substituted by all possible constants, either positive or negative
    
    • Negative atoms can also be simply left out
6.1 Example

- A total of 16 possible interpretations
  - \{\text{green(Hektor)}, \text{green(Christoph)}, \text{frog(Hektor)}, \text{frog(Christoph)}\}
  - \{\text{green(Hektor)}, \text{green(Christoph)}, \text{frog(Hektor)}\}
  - \{\text{green(Hektor)}, \text{frog(Christoph)}\}
  - \{\text{green(Christoph)}, \text{frog(Christoph)}\}
  - \ldots
6.1 Example

• But which of them are models for our program?
  – We have the fact green(Hektor).
    • Hence all models have to contain green(Hektor)
  – And we have the rule frog(X) :- green(X).
    • Hence all models also have to contain frog(Hektor)
  – But our program is not adversely affected by the atoms green(Christoph) and frog(Christoph)
    • On the other hand our models don’t need these atoms…
Thus, there are **3 models** for our program

- \{\text{green}(\text{Hektor}), \text{frog}(\text{Hektor})\}
- \{\text{green}(\text{Hektor}), \text{frog}(\text{Hektor}), \text{frog}(\text{Christoph})\}
- \{\text{green}(\text{Hektor}), \text{frog}(\text{Hektor}), \text{green}(\text{Christoph}), \text{frog}(\text{Christoph})\}

**Which one do we want?!**

- Note: the first model is the intersection of all models!
6.1 Herbrand Models

• The intended Herbrand Model for Datalog\(^f\) programs can thus be described as

  – **The least model**: A given Herbrand model \( M \) is called **least model**, iff \( M \subseteq M' \) for all other Herbrand models \( M' \)

    • The semantics induced by the least model is often called **stable model semantics**
    • Since negation is prohibited in Datalog\(^f\), there **exists always** a least model
6.1 Herbrand Models

- **Lemma**: Given a Datalog\(^f\) program \(\mathcal{P}\) and the set \(\mathcal{M}\) of all its Herbrand models. Then the least Herbrand model of \(\mathcal{P}\) is defined as \(M_\mathcal{P} := \bigcap \mathcal{M}\)

- \(M_\mathcal{P}\) represents the intended semantics of \(\mathcal{P}\), as it evaluates all given facts and rules to true, **but not more**

- Only what is **explicitly stated** by the program is true, the rest is considered false
• Consider **Datalog\textsuperscript{neg}** for a moment…
  
  – Given the constant Hector and the program: \( \text{toad}(X) :\neg \text{frog}(X) \).
  
  – We can come up with **two models** \{\text{toad}(Hector)\} and \{\text{frog}(Hector)\}
    
    • Both satisfy the program, but their intersection is empty…
    
    • Note that \((\neg A \rightarrow B)\) is equivalent to \((A \lor B)\)
Thus, for evaluating the semantics of a given Datalog program $\mathcal{P}$, actually finding the least Herbrand Model $\mathcal{M}_\mathcal{P}$ is essential.

– Unfortunately, finding $\mathcal{M}_\mathcal{P}$ using the intersections of models in the previous lemma is often not possible, because $\mathcal{M}$ may be of infinite size.
For more expressive logic languages (like Prolog), **deductive systems** are used to find the truth values for the elements of the Herbrand universe

- E.g., **SDL resolution**
- But this may lead to severe performance penalties

In Datalog, the problem is solved using the simpler **fixpoint iteration**

- A sound and complete deductive system for Datalog
6.2 Operative Semantics

• Basic idea of **fix point iteration**
  – Start with an empty subset $I_0$ of the Herbrand base of the logic language used by $\mathcal{P}$
    • Later, this subset will be identified with a special Herbrand interpretation, i.e. all atoms of the Herbrand base $B_\mathcal{L}$ evaluate to false
  – Transform the set $I_n$ into the set $I_{n+1}$, i.e. $I_{n+1} := T_\mathcal{P}(I_n) := T_\mathcal{P}^{n+1}(I_0) := T_\mathcal{P}(...T_\mathcal{P}(T_\mathcal{P}(I_0)))$
    • $T_\mathcal{P}$ is some transformation rule
6.2 Operative Semantics

• **Elementary Production** $T_P$
  
  $T_P : 2^{B_L} \rightarrow 2^{B_L}$
  
  • Maps an element of the power set of the Herbrand base to another, i.e. one *subset of atoms* to another subset of atoms

  $T_P : I \mapsto \{ B \in B_L | t \}$

  • here exists a *ground instance*

  $B ::= A_1, A_2, \ldots, A_n$ of a program clause such that $\{ A_1, A_2, \ldots, A_n \} \subseteq I$

  • Captures the idea of *forward-chaining*, i.e. start with base facts and produce new facts by applying the rules
– Remember: **ground instances** are quantifier-free subformulas of a formula in prenex form where all free variables are substituted with some term from the universe

– Example with $U_L = \{v_1, v_2, v_3, ...\}$

  • program clause: $\text{path}(X, Y) :- \text{edge}(X, Y)$.
    meaning $\forall X, Y \ (\text{path}(X, Y) \lor \neg \text{edge}(X, Y))$

  • A ground instance is $\text{path}(v_1, v_2) \lor \neg \text{edge}(v_1, v_2)$ with substitution $X|_{v_1}$ and $Y|_{v_2}$
6.2 Example

- Example program $\mathcal{P}$
  - $\text{edge}(v_1, v_2)$.
  - $\text{edge}(v_1, v_3)$.
  - $\text{edge}(v_2, v_4)$.
  - $\text{edge}(v_3, v_4)$.
  - $\text{path}(X, Y) :\! -\! \text{edge}(X, Y)$. [rule 1]
  - $\text{path}(X, Y) :\! -\! \text{edge}(X, Z), \text{path}(Z, Y)$. [rule 2]
6.2 Example

• Fixpoint Iteration:
  
  – $I_0 := \emptyset$
  
  – $I_1 := T_P(I_0) = \{\text{edge}(v1, v2), \text{edge}(v1, v3), \text{edge}(v2, v4), \text{edge}(v3, v4)\}$

  • The (empty) premises are triggered by the transformation rule
  
  • The elements of $I_1$ are the ground facts
6.2 Example

\[ I_2 := T_P(I_1) = I_1 \cup \{\text{path}(v1, v2), \text{path}(v1, v3), \text{path}(v2, v4), \text{path}(v3, v4)\} \]

- Rule 1: \(\text{path}(X, Y) :- \text{edge}(X, Y)\) is applied to all atoms in \(I_1\)
- Rule 2: \(\text{path}(X, Y) :- \text{edge}(X, Z), \text{path}(Z, Y)\) is **not** triggered, since there are no path-atoms in \(I_1\)

\[ I_3 := T_P(I_2) = I_2 \cup \{\text{path}(v1, v4)\} \]

- Rule 2: \(\text{path}(X, Y) :- \text{edge}(X, Z), \text{path}(Z, Y)\) is triggered

\[ I_4 := T_P(I_3) = I_3 \cup \{\} \]

- Fixpoint reached
- Set remains stable
With $T_P$ as constructed above it can be shown that...

- $I_n \subseteq I_{n+1}$, i.e. within each iteration, the set may only **grow** (the evaluation is monotonic)
- There exists an $f \geq 0$ such that $\forall m \geq f : I_m = I_{m+1}$
  - $f$ is called the **fixpoint**, after the fixpoint the sets are **stable**
  - Also, the following holds: $\forall m < f : I_m \subset I_{m+1}$
- $I_f$ can be identified with the **least Herbrand model** $M_P$
  - $I_f$ is not just some set of Herbrand base elements, but can be seen as a minimal interpretation that is consistent with the program $P$ (and thus a model)
6.2 Fixpoint Semantics

• Fixpoint iteration may be understood as a 
deductive system
  – The program $\mathcal{P}$ provides the axioms
  – The only deduction rule is $T_\mathcal{P}$

• Thus, fixpoint iteration purely syntactically 
produces inferred ground facts with each 
iteration
  – Inferred ground fact $W : \mathcal{P} \vdash W$
    • Either $W \in \mathcal{P}$ or $W$ can be obtained after a finite number of 
      iteration steps
Thus, for each inferred ground fact \( W \), a **proof tree** can be constructed:

– Proof tree has two types of nodes:
  
  • **Fact nodes**: Contains a inferred ground fact or a fact clause from \( \mathcal{P} \)
  
  • **Rule nodes**: Contains a rule from \( \mathcal{P} \)
6.2 Fixpoint Semantics

– The proof tree shows the minimal set of \textbf{rules} and \textbf{facts} that have been \textit{necessary to infer} \( W \)

  • \( W \) itself is in the tree root which is a fact node
  • Each level of the tree represents an iteration
  • The lowest level represents the first iteration
  • The depth of the tree thus represents the number of necessary iterations to deduce \( W \)
  • If the same clause is used multiple times, it is copied
6.2 Fixpoint Semantics

- Example (cont.): Proof tree of path(v1, v4)

- Alternative tree
• **Soundness Theorem:**
  – Each inferred ground fact \( W \) with \( \mathcal{P} \vdash W \) is also a **semantic conclusion** \( \mathcal{P} \models W \)

• **Proof:**
  – **Idea:** Show by induction over the depth of the proof tree of \( W \)
  
  – **Induction Base:**
    • If \( W \) has a proof tree of depth 0 then \( W \in \mathcal{P} \). Thus \( W \) must be in each Herbrand model and \( \mathcal{P} \models W \).
• **Induction Step:**

  – Assume $W$ has a proof tree of depth $i+1$. Then there is a rule $R \equiv B \vdash A_1, \ldots, A_n$ and some ground facts $F_1, \ldots, F_n$ at level $i$ such that $W$ can be inferred in one step by applying $T_P$ on $R$ and $F_1, \ldots, F_n$.

  – Since the facts $F_1, \ldots, F_n$ appear on level $i$, they each must have a proof tree of a depth $\leq i$.

  – By induction hypothesis, we have for $i \leq k \leq n : P \models F_k$; thus for each Herbrand model $I$ we have $F_k \in I$.

  – Since also $R \in I$, we also have $W \in I$ and thus $P \models W$.
• **Completeness Theorem:**
  – Each semantic conclusion $\mathcal{P} \models W$ is also an inferred ground fact $W$ with $\mathcal{P} \vdash W$

• **Proof:**
  – Consider the set $\text{infer}(\mathcal{P}) := \{W \mid W \text{ is ground fact and } \mathcal{P} \vdash W\}$
  – By definition of $\vdash$, each fact $W \in \mathcal{P}$ is also in $\text{infer}(\mathcal{P})$
  – Consider any rule $R \in \mathcal{P} \equiv B \vdash A_1, \ldots, A_n$
  – Assume a substitution $\rho$ such that $\mathcal{P} \vdash \rho(A_1), \ldots, \mathcal{P} \vdash \rho(A_n)$
Then, also $\rho(B) \in \text{infer}(\mathcal{P})$ and $\mathcal{P} \vdash \rho(B)$

Hence, $\text{infer}(\mathcal{P})$ is a Herbrand model of $\mathcal{P}$

Now assume that $\mathcal{P} \models W$

Thus, $W$ is in each Herbrand model of $\mathcal{P}$ and particularly in $\text{infer}(\mathcal{P})$, so finally $\mathcal{P} \vdash W$

Finally, we may combine both theorems:

Given a set of Datalog clauses $\mathcal{P}$, then $\mathcal{P} \models W$, iff $\mathcal{P} \vdash W$
• Corollary: If $\mathcal{P}$ is finite, then also $\{W \mid W \text{ is ground fact and } \mathcal{P} \vdash W\}$ is finite
  
  – Thus, any Datalog model can be represented and computed in finite space and time

  – By proving this, we can show that there has to be a fixpoint which can be reached
6.2 Fixpoint Semantics

• Naïve algorithm for query execution:
  – Given a Datalog program $\mathcal{P}$ and a query $Q \equiv A_1, \ldots, A_n$
  – Start **fixpoint iteration** on $\mathcal{P}$
    • As soon as $\mathcal{P} \vdash \neg Q$, the query is **unsatisfiable** and return an empty result set
    • For every inferred ground fact $W$ which is a ground instance of $Q$, put $W$ into the result set
    • If fixpoint is reached, return result set
  – Please note: in every iteration step, the whole set of currently known ground facts is also **recomputed**!
6.2 Example

- Example program $\mathcal{P}$
  - $e(1, 2)$.
  - $e(1, 3)$.
  - $e(2, 4)$.
  - $e(3, 4)$.
  - $e(4, 5)$.
  - $p(X, Y) :- e(X, Y)$. [rule 1]
  - $p(X, Y) :- e(X, Z), p(Z, Y)$. [rule 2]
6.2 Example

- **Query**: $p(2, X)$?
  - (i.e. which vertices can be reached starting from 2)

- **Fixpoint-Iteration**
  - $I_0 := \{\}$
  - $I_1 := I_0 \cup \{e(1,2), e(1,3), e(2,4), e(3,4), e(4,5)\}$
  - $I_2 := I_1 \cup \{p(1,2), p(1,3), \fbox{p(2,4)}, p(3,4), p(4,5)\}$
  - $I_3 := I_2 \cup \{p(1,4), \fbox{p(2,5)}, p(3,5)\}$
  - $I_4 := I_3 \cup \{p(1,5)\}$
  - $I_5 := I_4 \cup \{\}$
  - Result := $\{p(2,4), p(2,5)\}$
6.2 Expressiveness

• Thus, Datalog$^f$ has a clear **operative semantics** which allows computation of models and facts
  – But, remember, we excluded negation… but do we need it?

• Theorem: **Datalog$^f$ is Turing-complete**
  – Thus, by using **Datalog$^f$** you can compute anything you can compute with any other programming language (like C, Java, Pascal, etc)
• However, it might be a nice feature to be able to express negation
  – e.g. “A day which is no holiday and no week end is a working day.“
  – workday(X) :-
    day(X), not holiday(X), not weekend(X)
  – Negation allows for a more intuitive modeling of the real world in Datalog programs
However, allowing negation opens up many problems

- For Datalog\(^f\), we used the notion of the least Herbrand model
- Then the **least Herbrand** model of \(\mathcal{P}\) can be defined as \(M_\mathcal{P} := \bigcap \mathcal{M}\) \((\mathcal{M}\) being any model of \(\mathcal{P}\))
- The previous definition was used to prove the **soundness and completeness** of the fixpoint iteration.
6.2 Datalog\textsuperscript{neg}

• Lets go back some slides
  – Given the constant Hector and the program: toad(X) :- not frog(X).
  – We can come up with two models \{toad(Hector)\} and \{frog(Hector)\}
    • Both satisfy the program, but their intersection is empty…
    • Note that (¬ A → B) is equivalent to (A ∨ B)
    • Thus, there is no least Herbrand model and fixpoint iteration is broken
6.2 Datalog$^{\text{neg}}$

• Usually, Datalog$^{\text{neg}}$ don’t have a least Herbrand model, instead they may have multiple minimal models

  – **Minimal Model**: A given Herbrand model $M$ is **minimal** iff there is no other Herbrand model $M'$ such that $M' \subset M$

    • The induced semantics is called **minimal Herbrand semantics**
• However, it is **unclear** which model should be used
  – Both \{toad(Hector)\} and \{frog(Hector)\} are valid **minimal models** of the previous example

• So, we need a **deterministic decision criteria** for selecting an **appropriate model**
• One promising way is to use the results of stratification
• **Remember:** Stratification tries to determine negative dependencies of predicates within a program by ordering predicates into an hierarchy
  – Only programs that can be stratified can also be executed
6.2 Datalog\textsuperscript{neg}

• Example:

- \text{path}(1,2). \text{path}(1,3). \text{path}(3,4). \text{toll}(1,2).
  \text{goodpath}(X, Y) \text{ :- } \text{path}(X, Y), \neg \text{toll}(X).
  \text{goodpath}(X, Z) \text{ :- } \text{goodpath}(X, Y), \text{goodpath}(Y, Z).

- $S_1 := \{\text{def(path)}, \text{def(toll)}\}$ : first stratum
  $S_2 := \{\text{def(goodpath)}\}$ : second stratum

All defining clauses for the given predicate (i.e. with the predicate in the head)
• Problems with negation in detail
  – toad(y) :- ¬frog(y)
  – To evaluate this rule, all universe ground terms have to be tested (in some cases of Datalog, the universe may be infinite…)
  – It’s possible that just for a small (finite) part of the universe, frog(y) is true
    • an excessively large or even infinite number of toad facts have to be included in the model
    • This rule is thus unsafe (possibly infinite or excessively large models)
– Furthermore, the choice of models is ambiguous

  • No information about frog(Hector)… does this mean that toad(Hector) is true? Or might frog(Hector) be true although it was not stated explicitly?
These two problems (ambiguous, unsafe) can be countered with the following constraints:

– “If a variable appears in a negative literal, it must also appear in a positive literal in the body.”

– `toad(y) :- green(y), ¬frog(y)`
  
  • Variable `y` also appears in a positive literal (`y` is “grounded”)
  • This is called “safe negation”
Now, we can first evaluate the positive and then the negative literals

- For any $y$ for which $\text{toad}(y)$ becomes true, $\text{green}(y)$ needs to be true first
- If that was the case, there has been a rule/ fact stating this; i.e. number of candidate terms is very limited
- To capture this, organize evaluation **strata per strata**
  - Positive facts are in lower strata
  - i.e.: to fire rule, positive literal has to be fact in a higher strata, negative literal must not be a fact of a higher strata
6.2 Datalog\textsuperscript{neg}

• In the following, we formalize the observation of the previous slides
• Based on the **negative dependencies**, we can define a **Priority Relation** \(<_p\) on the elements of the Herbrand base \(B_L\)
  
  \[
  P(t_1, \ldots, t_n) <_p Q(s_1, \ldots, t_m) \text{ iff there is a negative edge from } P \text{ to } Q \text{ in the program connection graph (PCG) of } P
  \]

• **Lemma**: If a program \(P\) is **stratified**, \(<_p\) is an **irreflexible partial order**
  
  \[
  \text{If not, there may be cycles, e.g. } \begin{align*}
  P(t_1, \ldots, t_n) <_p Q(s_1, \ldots, t_m) <_p P(t_1, \ldots, t_n)
  \end{align*}
  \]
6.2 Datalog\textsuperscript{neg}

• Based on the priority relation, we define a preference relation between minimal models
  – Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be models of $\mathcal{P}$
  – Then $\mathcal{M}_1$ is preferred over $\mathcal{M}_2$ ($\mathcal{M}_1 \leq \mathcal{M}_2$) iff
    • $\mathcal{M}_1 = \mathcal{M}_2$
    • OR $\mathcal{M}_1 \neq \mathcal{M}_2$ and for all $A \in \mathcal{M}_1 \setminus \mathcal{M}_2$ there exists a $B \in \mathcal{M}_2 \setminus \mathcal{M}_1$ such that $A <_p B$

• A model $\mathcal{M}$ is called perfect model iff
  – $\mathcal{M} \leq \mathcal{M}'$ for all Herbrand Models $\mathcal{M}'$ of $\mathcal{P}$
**Example:**

- \( \mathcal{P} \) :
  - \( \text{green(Hector)} \)
  - \( \text{toad(X)} :: \text{green(X)}, \text{not frog(X)}. \)

- **Two models:**
  - \( \mathcal{M}_1 := \{ \text{toad(Hector), green(Hector)} \} \)
  - \( \mathcal{M}_2 := \{ \text{frog(Hector), green(Hector)} \} \)

- \( \text{toad(Hector)} \prec_{\mathcal{P}} \text{frog(Hector)} \)
  \[ \Rightarrow \mathcal{M}_1 \leq \mathcal{M}_2 \text{ and } \mathcal{M}_1 \text{ is the perfect model} \]
• **Theorem**: For each Datalog\(^\text{neg}\) program, there exists a **perfect model** which is also a **minimal model**
  
  – We define that this **perfect** and **minimal** model is the **intended semantic** of a given Datalog\(^\text{neg}\) program

• However, **fixpoint iteration** is still broken and needs some modification
  
  – Idea: Modify elementary production rule \(T_\mathcal{P}\) such that it works along the strata of \(\mathcal{P}\)
  
  – Negation as failure semantics should be captured
• Elementary Production \( T^J \) depending on \( J \)
  
  \[- T^J : 2^{B_L} \rightarrow 2^{B_L} \]
  
  \[- T^J : I \mapsto \{ B \in B_L \mid \text{there exists a ground instance} \]
  
  \[ B : A_1, ..., A_n, \neg C_1, ..., \neg C_m \text{ of a program} \]
  
  \[ \text{clause such that } \{A_1, A_2, ..., A_n\} \subseteq I \]
  
  \[ \text{and for all } 1 \leq i \leq m : C_1 \notin J \} \]
• Example:

- goodpath(X, Y) :- path(X, Y), ¬toll(X).
  goodpath(X, Z) :- goodpath(X, Y), goodpath(Y, Z).
- \( J := \{\text{toll}(1,2)\} \)
  \( I := \{\text{path}(1,2), \text{path}(2,4), \text{path}(1,4)\} \)
- \( T^J_p(I) := I \cup \{\text{goodpath}(2,4), \text{goodpath}(1,4)\} \)
Based on the new definition of **elementary production**, we can define a new **iterated fixpoint iteration** using **stratification**

- Let $\mathcal{P}$ be a stratified program as $\mathcal{P} := \mathcal{P}_0 \cup \ldots \cup \mathcal{P}_n$

- $I_0 := T_{\mathcal{P}_0}^\infty (\emptyset)$
- $I_1 := T_{\mathcal{P}_1 \cup \mathcal{P}_0}^\infty (T_{I_0}^{\mathcal{P}_1 \cup I_0})$
- ...
- $I_n := T_{\mathcal{P}_n \cup \ldots \mathcal{P}_0}^\infty (T_{I_{n-1}}^{\mathcal{P}_n \cup I_{n-1}})$

- As soon as we reach the fixpoint $n$, we call $I_n$ the **iterated fixpoint** of $\mathcal{P}$
Informally, this means

- Partitioned all clauses of the program $\mathcal{P}$ such that each partition corresponds with a strata
- Apply $T_{\text{neg}, \mathcal{P}}$ to the lowest strata program fragment
  - This creates a set of all
6.2 Operative Datalog\textsuperscript{neg}

• Theorem: The iterated fixpoint $I_n$ is indeed the minimal perfect Herbrand model of $\mathcal{P}$
  – Thus, the iterated fixpoint iteration provides a computable operative semantic for Datalog\textsuperscript{neg}

• However, the \textbf{performance} of naïve operational semantics of Datalog\textsuperscript{neg} or Datalog\textsuperscript{f} still remains suboptimal
  – Room for further improvement $\rightarrow$ next lecture
6.2 Operative Datalog\textsuperscript{neg}

• Example:

\[ P_1 := \{ \]
\quad edge(1,2). edge(1,4). edge(2,4).
\quad edge(3,4). toll(1,2).
\quad path(X, Y) :- edge(X, Y).
\quad path(X, Y) :- edge(X, Z), path(Z, Y) \}

\[ P_2 := \{ \text{goodpath}(X, Y) :- path(X, Y), \neg \text{toll}(X). \]
\quad \text{goodpath}(X, Z) :- \text{goodpath}(X, Y), \text{goodpath}(Y, Z) \} \]
6.2 Operative Datalog$^{\text{neg}}$

- $I_1 := T_\mathcal{P}^\infty(T_{\text{neg}}, \mathcal{P}_1) = \\
  \{\text{edge}(1,2), \text{edge}(1,4), \text{edge}(2,4), \text{edge}(3,4), \text{toll}(1.2), \text{path}(1,2), \text{path}(1,3), \text{path}(1,4), \text{path}(2,4), \text{path}(3,4)\}$

- $I_2 := T_\mathcal{P}^\infty(T_{I_0, \text{neg}}^{I_1}, \mathcal{P}_2 \cup I_1) = I_1 \cup \\
  \{\text{goodpath}(1,3), \text{goodpath}(2,4), \text{goodpath}(3,4), \text{goodpath}(1,4)\}$
During the 80ties, commonly five computer generations have been distinguished

- **0th generation**: Full mechanical (like IBM407) or mechanical switching computers (like Harvard Mark I)

- **1st generation**: (around 1940’s) using pluggable vacuum tubes (ENIAC)
6.3 5th Generation Computer

- **2nd generation**: (after 1953) computers using transistors instead of vacuum tubes (like Manchester Mark I or IBM 7090)

- **3rd generation**: (around 1964)
  Usage of integrated circuits
  - Large number of transistors on a single chip
  - IBM 360

- **4th generation**: Microprocessors
  - Entire processing units on a single chip
  - The F-14A “Tom Cat” Microprocessor, Intel 4004
In the mid-70’s, Japan felt that it was vastly behind the U.S. and the U.K. regarding computer technology.

To fight this fact, the Ministry of International Trade and Industry (MITI) requested a roadmap of potential future “hot topics” from Japanese research companies and academics.

– Mainly Japan Information Processing Development Center (JIPDEC)
• 5 potential fields have been identified
  – Inference computer technologies for knowledge processing
  – Computer technologies to process large-scale databases and knowledge bases
  – High performance workstations
  – Distributed functional computer technologies
  – Super-computers for scientific computing
Based on these, the 5th Generation Computer (FGCS) project was funded for a 10-year-period

- Started 1982 with a funding of 900 Million US-$
**6.3 5th Generation Computer**

- **Idea:** Build a computer which is **completely different** from current systems
  - It runs on top of a massive distributed **knowledge base**
  - It uses **logic programming** only
  - It allows for massively **distributed processing** of logical inference
    - 100M-1G LIPS compared to “normal mainframe” 100K LIPS
    - LIPS: Logical Inference Per Second
After the **striking results** of the previous MITI projects (e.g. **consumer electronics** in the 70ties and **automotives** in the 80ties), the other active countries in computer research were **struck with fear**.
• **Counter projects worldwide (Sputnik Effect):**
  
  – Microelectronics and Computer Technology Corporation (MCC) in US
  – Alvey in UK
  – ESPRIT and ECRC in Europe
  – …
6.3 5th Generation Computer

• Core results have been
  – Prototype Parallel Inference Machines PIM/m, PIM/p, PIM/I, PIM/k and PIM/c
  – Parallel Logic Programming Language KL-I
  – Parallel Logic Based Operation System PIMOS
Parallel DBMS Kappa-P
Theorem Prover MGTP
Inference Engine Quixote
Application Programs
• Legal Reasoning Systems, VLSI-CAD, Generic Information Processing, Software Generation, Expert Systems, etc.
Depending on whom you ask, the project was either a complete failure or just ahead of its time (and still a failure)

- Cheaper desktop system with standardized hardware developed much faster and became cheaper
  - No parallelization necessary
  - FGCS had no focus on HCI
- Logic Programming never took foot
- The A.I. winter killed a lot of A.I. dreams
In any case, many ideas currently return:

- Parallelization in massive multi-cores
- Reasoning in the form of Logic Programming for the Semantic Web
- Knowledge Based Systems
• Implementation of Datalog
• Efficient computation of fixpoints