Decomposition of Linear Recursive Logic Programs

Dietmar Seipel

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen *
Sand 13, D - 7400 Tübingen, West-Germany

Abstract

In practise, most recursive logic queries to a deductive database are expressed by linear recursive datalog programs with exactly one linear recursive rule, so-called **linear datalog sirups**.

The notion of **k-sided linear datalog sirups** has been introduced by Naughton, who characterized one-sided linear datalog sirups based on a graph model.
We use another graph representation of linear datalog sirups for extending this characterization to arbitrary sidedness k. It is shown that $0 \leq k \leq n'$, where n' is the dimension of the sirup, i.e. the arity of its recursive predicate symbol. An efficient (quadratic time) algorithm for the determination of the sidedness k is presented.

We will define the **canonical k-sided normal form (CKNF)** for linear datalog sirups. Every k-sided linear datalog sirup can be normalized to an equivalent k-sided CKNF sirup. Every k-sided CKNF sirup can be **decomposed** by **counting techniques** into k lower-dimensional linear sirups in a very simple normal form for non-datalog sirups, the generalized form for transitive closure, which allows for the usage of very efficient query evaluation algorithms.

1 Introduction

For supporting artificial intelligence and automated engineering design by relational databases, it is necessary to augment relational query languages by some kind of recursive facilities. Section 1.1 will give a definition of a **deductive database**, which includes **logic programs** for formulating recursive queries.

In the last years, there has been a lot of research on recursive query processing. A survey is given by [BaRa 86]. Three **characteristics** identified by [BaRa 86] and [HaHe 87] massively influence the **CPU-performance** and the **I/O** and **storage costs**:

*This work was done at the Universität Würzburg.

- The **set of relevant facts** is formed by those facts that transitively derive answers of the query. A common strategy is to incorporate **selections** given by the query to restrict query processing to the relevant facts and thus reduce the size of the intermediate database. The best-known algorithms are the **magic set** and **counting** methods for linear sirups, cf. [BaMaSaUl 86], and their generalizations for non-linear recursive logic programs, cf. [BeeRa 87], [SaZa 87].

- **The arity (dimension) of the intermediate relations** is very essential, cf. [HeNa 84], [BaMaSaUl 86], etc.. E.g. for a linear sirup, the complexity of evaluating a query grows **more than linear** with its dimension. **Counting methods** reduce the arities of general recursions by storing how the tuples were derived.

- The problem of reducing or eliminating the **duplication of work** is investigated in detail in [HaHe 87], where four different levels are examined.
  On the **precompilation level**, a logic program can be simplified based on its corresponding **rule/goal graph**, cf. [Ul 85], before it is evaluated against the extensional database.
  On the **iteration level**, the most common techniques are performing **selections first** to reduce the size of the intermediate database, cf. [AhUl 79], [KiLo 86], [AgDe 88], using **frontier relations** to avoid the repeated processing of subexpressions of different compiled exit rules like the **wavefront techniques**.
  On the **tuple processing level**, excluding processed facts from repeated processing avoids duplicate computations and removes redundancy arising from 'asynchronous' and 'cyclic' databases, cf. [BaRa 86], [GüKiBa 87], [HaNa 88], [Sei 88].
  On the **file accessing level**, there are a lot of methods for minimizing the **I/O costs**.

The **counting** technique, which will be described in **section 1.2**, uses most of the techniques just mentioned. Like most known methods, it efficiently works only for special application domains. The problem of supporting arbitrary, complex recursive queries is not yet solved completely.

There is a number of **graph-based optimization techniques** that consider special properties of the linear recursive rule r of a linear datalog sirup LP:

- The **canonical form for transitive closure (CFTC)** is a normal form for linear datalog sirups introduced by Jagadish, Agrawal and Ness in [JaAgNe 87]. They show by the help of their **substitution graphs** that every linear datalog sirup can be normalized to an equivalent CFTC sirup.

- J. Naughton investigated linear datalog sirups without constants or repeated patterns in the consequent of the linear recursive rule r based on his **full argument-variable graphs**.
  [Na 87] defined the concept of **k-sided recursions** and gave a characterization of one-sided linear datalog sirups. He argued that queries w.r.t. one-sided linear sirups can be evaluated efficiently. There are efficient methods for treating two-sided linear sirups as well, c.f. [HaHe 87], [HaNa 88], etc.. But, up to now, no complete

characterization of k-sided linear datalog sirups has been given for $k \geq 2$.
In [Na 89], an algorithm is introduced for detecting and removing **redundant atoms** from bounded linear datalog sirups.

- [Io 85] introduces a graph representation of r, which is called the **Ioanidis-graph (I-graph)** in [YoHeHa 88]. [IoWo 86] shows that certain properties of the I-graphs allow a reduction of the dimension of the linear sirup.

- In [YoHeHa 88], **strongly stable**, linear datalog sirups are considered as desirable. They are characterized by 'independent, one-directional cycles' in the I-graphs. The optimization of non-strongly stable linear sirups, which is mentioned as an open problem there, can be solved by the decomposition approach presented here.

Section 2 investigates the very general class of datalog sirups in **standard linear form (SLF)** by a graph-model, that extends the substitution graphs to **augmented substitution graphs**. It turns out that every SLF sirup is k-sided for some $k \in \mathbb{N}_0$, which can be determined by an efficient algorithm.

Section 3 yields a generalization of the counting method to the **canonical k-sided normal form (CKNF)** introduced here. It is shown that a k-sided SLF sirup can be normalized to an equivalent k-sided CKNF sirup. A k-sided CKNF sirup can be decomposed by **counting techniques** into k 'less complex' linear sirups in a special normal form for non-datalog sirups, the **generalized form for transitive closure**, for which very efficient query evaluation algorithms are available.

1.1 Basic Definitions and Notations

Syntactically, a **deductive database** $DDB = (DB, LP)$ is given by a set DB of facts, which resemble a relational database, and a logic program LP, which consists of a set of rules for deriving new facts from DB, cf. [Lo 84]:

(i) *A* **fact** *is an atom $p(t_1, \ldots, t_n)$, where p is a predicate symbol, $n \in \mathbb{N}$, and t_i ($1 \leq i \leq n$) is a term over constant symbols and function symbols.*
A **goal** *is given by a sequence of atoms.*
A **rule** *r is given by*

$$p_0(X_0) : -p_1(X_1), \ldots, p_n(X_n).$$

where $n \in \mathbb{N}$; the atom $p_0(X_0)$ is its **head** *and the goal $p_1(X_1), \ldots, p_n(X_n)$ is its* **body**.

(ii) *Given a predicate symbol p.*
p is **intensional**, *iff it occurs in the head of a rule $r \in LP$; otherwise it is* **extensional**.
DB is **extensional**, *iff it contains only facts with extensional predicate symbols.*
*The p-***database** *of DB contains all facts $p(t_1, \ldots, t_n) \in DB$.*

(iii) *A rule r is a* **datalog rule**, *iff it contains no function symbols in the terms of its atoms. LP is a* **datalog program**, *iff all rules $r \in LP$ are datalog rules.*

The **semantics** of a deductive database $DDB = (DB, LP)$ is defined according to the **bottom-up** or **naive** evaluation algorithms, cf. [BaRa 86]:

(i) *The database containing all facts* $p(t_1, \ldots, t_n)$ *that can be derived from* DB *by repeated applications of rules* $r \in LP$ *is denoted by* $LP^\omega(DB)$.

Y. Sagiv introduces two notions of **equivalence** for two logic programs LP_1, LP_2 with the same extensional predicate symbols, cf. [Sa 87], where LP_1 and LP_2 are called

(i) **equivalent**, *iff* $LP_1^\omega(EDB) = LP_2^\omega(EDB)$ *for all extensional databases* EDB.

(ii) p**-equivalent** *for a predicate symbol* p, *iff the* p*-databases of* $LP_1^\omega(EDB)$ *and* $LP_2^\omega(EDB)$ *are equal for all extensional databases* EDB.

(iii) **uniformly equivalent**, *iff* $LP_1^\omega(DB) = LP_2^\omega(DB)$ *for all databases* DB.

The **equivalence** problem for two general logic programs is **undecidable**, but the problem of **uniform equivalence** is **decidable**.

The **complexity** of evaluating recursive queries mainly depends on the structure of the **recursive rules**, cf. [BaRa 86], [YoHeHa 88]. Consider a logic program LP, a rule $r \in LP$ and two predicate symbols p and q:

(i) q **derives** p, *iff there is a* p*-rule* $r \in LP$ *whose body contains an atom* $q(X)$.
q **transitively derives** p, *iff* q *derives* p *or* q *derives a predicate symbol* q' *that transitively derives* p.

(ii) r *is* **recursive**, *iff its head predicate symbol transitively derives one of its body predicate symbols.*

(iii) LP *is a* **linear single recursive rule program (linear sirup)**, *iff it contains exactly one* **linear recursive rule** r *of the form*

$$p(X) :- p(U), a_1(U_1), \ldots, a_n(U_n).$$

and some non-recursive rules, the so-called **exit rules**. *The* **dimension** *of the linear sirup* LP *is given by the arity of the* **recursive predicate symbol** p.

E.g. the **deductive database** $DDB = (DB, LP)$ given by

- the **parentship database**
  $DB = \{\ par(cain, adam),\ par(abel, adam),\ par(cain, eve),$
  $par(abel, eve),\ par(sem, abel)\ \}$

- and the 2-dimensional **linear datalog sirup** $LP = \{\ r, e\ \}$,

$$\begin{array}{lllll} r: & anc(x_1, x_2) & :- & anc(x_1, u_2), par(u_2, x_2). \\ e: & anc(x_1, x_2) & :- & par(x_1, x_2). \end{array}$$

which specifies that x_2 is an **ancestor** of x_1, iff
u_2 is an ancestor of x_1 and x_2 is a parent of u_2 (cf. the linear recursive rule r), or
x_2 is a parent of x_1 (cf. the non-recursive exit rule e),

defines the **ancestorship database** $LP^{\omega}(DB)$.

We need the following **normal forms** for linear sirups. The canonical form for transitive closure is defined in [JaAgNe 87]. The generalized form for transitive closure (for non-datalog sirups) is derived from the generalized composition operators of [SiSo 88]:

(i) *A linear datalog sirup LP is in* **canonical form for transitive closure (CFTC)**, *iff in its linear recursive rule*

$$r: \ p(X) :- p(U), a_1(U_1), \ldots, a_n(U_n).$$

the argument vectors X and U are **isomorphic**, *i.e. any common arguments are in the same positions in both vectors and any non-common arguments are variables, and the pattern of repetition is the same in both vectors.*

(ii) *A linear sirup LP is in* **generalized form for transitive closure (GFTC)**, *iff in its recursive rule*

$$r: \ p(X, Y^0, Y^2, Z^0, m+1) :- p(X, Y^0, Y^1, Z^0, m), a(X, Y^1, Y^2, Z^1, V^2).$$

the argument vectors $X, Y^0, Y^1, Y^2, Z^0, Z^1, V^2$ are disjoint and
the argument vectors Y^0, Y^1, Y^2 and Z^0, Z^1 have the same length, respectively.
Here, the **concatenation** *of several argument vectors $W_i = (t_{i,1}, \ldots, t_{i,m_i})$*
($1 \leq i \leq n$) is denoted by $(W_1, \ldots, W_n) = (t_{1,1}, \ldots, t_{1,m_1}, \ldots, t_{n,1}, \ldots, t_{n,m_n})$.

1.2 The Counting Technique

The common bottom-up evaluation of a linear sirup LP iteratively increases an intermediate p-database, where all a_i-databases are relevant for deriving new facts in each iteration.

In comparison to that, [JaAgNe 87] suggest to transform a CFTC sirup LP to an equivalent set of independent linear sirups, each depending only on one a_i-database. Unfortunately, there are CFTC sirups which are not decomposable in this way, e.g. $LP = \{r, e\}$:

$$r: \ p(x_1, x_2, x_3) \ :- \ p(u_1, u_2, u_3), a_1(x_1, u_1, x_2), a_2(x_3, u_3, u_2).$$

But the **counting method** of [BaMaSaUl 86] shows that the intended decomposition is possible for n-dimensional linear datalog sirup $LP = \{r, e\}$ in **linear variable pattern**:

$$\begin{aligned} r: \ & p(x_1, \ldots, x_n) \ :- \ p(u_1, \ldots, u_n), a_1(x_1, u_1), \ldots, a_n(x_n, u_n). \\ e: \ & p(x_1, \ldots, x_n) \ :- \ a_0(x_1, \ldots, x_n). \end{aligned}$$

The resulting linear sirups LP_i are 3-dimensional GFTC sirups.

The decomposition can be derived from the compiled exit rules e^m ($m \in \mathbb{N}_0$), which form an equivalent, infinite, non-recursive logic program $LP^\omega = \{e^m | m \in \mathbb{N}_0\}$, where u_i^j ($1 \leq i \leq n$, $j \in \mathbb{N}_+$) are different variable symbols:

$$\begin{aligned} e^m : \; p(x_1, \ldots, x_n) \; :- \; & a_1(x_1, u_1^1), a_1(u_1^1, u_1^2), \ldots, a_1(u_1^{m-1}, u_1^m), \ldots, \\ & a_n(x_n, u_n^1), a_n(u_n^1, u_n^2), \ldots, a_n(u_n^{m-1}, u_n^m), a_0(u_1^m, \ldots, u_n^m). \end{aligned}$$

Thus, obviously the logic program $LP' = LP_1 \cup \ldots \cup LP_{n+1}$ is p-equivalent to LP. The subsets $LP_i = \{r_i, e_i\}$ ($1 \leq i \leq n$) can be evaluated independently on the a_i-databases DB_i. $LP_{n+1} = \{r_{n+1}, e\}$ combines the databases $LP_i^\omega(DB_i)$.

$$\begin{aligned} r_i : \; p_i(x_i, u_i^2, m+1) \; &:- \; p_i(x_i, u_i^1, m), a_i(u_i^1, u_i^2). \\ e_i : \; p_i(x_i, u_i^1, 1) \; &:- \; a_i(x_i, u_i^1). \\ r_{n+1} : \; p(x_1, \ldots, x_n) \; &:- \; a_0(u_1^1, \ldots, u_n^1), p_1(x_1, u_1^1, m) \ldots, p_n(x_n, u_n^1, m). \end{aligned}$$

One desirable property of a logic program is **safety**, which means that the bottom-up evaluation terminates with a finite database for all finite extensional databases.
LP' is not safe, since there exist so-called **cyclic** a_i-databases DB_i, where $LP_i^\omega(DB_i)$ is infinite.
For **acyclic** databases, the naive evaluation of LP' will always terminate and efficiently compute the desired result.
The **improved method** of [HaNa 88] for $n = 2$ is the best among all known methods which can handle **cyclic** databases DB_i as well. It has the same complexity $O(h \cdot f)$ as the counting methods, where h is the number of different constant symbols in $LP \cup DB$ and f is the number of facts in DB.

2 Characterization of k-Sided SLF Sirups

A certain subclass of linear datalog sirups, which we will call **standard linear sirups (SLF sirups)**, is frequently considered in literature.

Definition 2.1
A linear datalog sirup LP is in **standard linear form (SLF)**,
iff for its linear recursive rule

$$r : \; p(X) :- p(U), a_1(U_1), \ldots, a_n(U_n).$$

where $X = (x_1, \ldots, x_{n'})$, $U = (u_1, \ldots, u_{n'})$, n' is the dimension of LP, the following holds:

(SLF-1) *r is range-restricted,*
i.e. all variable symbols in its head occur in its body.

(SLF-2) *for all $1 \leq i \leq n'$: x_i is a variable, and*
for all $1 \leq i < j \leq n'$: x_i and x_j are different,
i.e. there are no constants and no repeated patterns in the consequent.

(SLF-3) *for $1 \leq i < j \leq n$: U_i and U_j share no common arguments.*

Firstly, the property **(SLF-1)** guarantees the safety of the datalog sirup. Secondly, the **sidedness** of an SLF sirup is defined in terms of a special **expansion** for linear datalog sirups in [Na 87], which requires the assumption **(SLF-2)**. According to [JaAgNe 87], every linear datalog sirup can be simplified to an equivalent linear datalog sirup having this property. Thirdly, a k-sided, n'-dimensional linear datalog sirup LP with the properties (SLF-1) and (SLF-2) can easily be transformed to an equivalent k-sided, n'-dimensional linear datalog sirup LP' having the property **(SLF-3)**, too.

2.1 Expansions and k-Sided Linear Sirups

The concept of **SLD-resolution** is introduced in [Lo 84]. [Na 87] describes a special version, where - given a logic program LP and a goal G_0 - a unique sequence $(G_i \mid i \in \mathbb{N}_0)$ of SLD-resolvents G_i is generated. This sequence is called the **expansion** of LP w.r.t. G_0.

Consider a linear datalog sirup LP with the property (SLF-2) and the recursive rule

$$r: \quad p(X) :- p(U), a_1(U_1), \ldots, a_n(U_n).$$

The 0-th resolvent is $G_0 = p(X) = p(U)\theta_0$. The i-th resolvent G_i ($i \in \mathbb{N}_0$) looks like

$$G_i = p(U)\theta_i, a_1(U_1)\theta_1, \ldots, a_1(U_1)\theta_i, \ldots, a_n(U_n)\theta_1, \ldots, a_n(U_n)\theta_i.$$

G_{i+1} is obtained by resolving G_i and r via the atoms $p(U)\theta_i$ and $p(X)$, respectively. Their **most general unifier (mgu)** *θ_{i+1} is chosen as a product $\theta_{i+1} = \theta_{i+1,1}\theta_{i+1,2}$, where $\theta_{i+1,1}$ is the (unique) mgu that superscribes every variable symbol in r by $i+1$, and $\theta_{i+1,2}$ is the (unique) mgu, such that $p(U)\theta_i = (p(X)\theta_{i+1,1})\theta_{i+1,2}$.*

The **SLD-resolvents** G_i, which are called **strings** in [Na 87], will be considered as **multisets**, i.e. double appearing atoms will be treated as different elements.

Example 2.2 *Consider the SLF sirup $LP = \{r, e\}$:*

$$\begin{array}{lrcl} r: & p(x_1,x_2,x_3,x_4,x_5,x_6) & :- & p(u_1,u_2,u_3,x_3,x_5,u_6), \\ & & & a_1(x_1,u_1), a_2(x_2,u_3), a_3(x_4,u_2), a_4(x_6). \\ e: & p(x_1,x_2,x_3,x_4,x_5,x_6) & :- & a_0(x_1,x_2,x_3,x_4,x_5,x_6). \end{array}$$

Then, the SLD-resolvents G_i ($i = 1, 2$) are given by

$$\begin{array}{lcll} G_1 & = & p(u_1^1,u_2^1,u_3^1,x_3,x_5,u_6^1), & a_1(x_1,u_1^1), a_2(x_2,u_3^1), a_3(x_4,u_2^1), a_4(x_6), \\ G_2 & = & p(u_1^2,u_2^2,u_3^2,u_3^1,x_5,u_6^2), & a_1(x_1,u_1^1), a_2(x_2,u_3^1), a_3(x_4,u_2^1), a_4(x_6), \\ & & & a_1(u_1^1,u_1^2), a_2(u_2^1,u_3^2), a_3(x_3,u_2^2), a_4(u_6^1). \end{array}$$

Continuing with **notations**, *the atoms $A_i^j = a_i(U_i)\theta_j$ ($1 \le i \le n$, $j \in \mathbb{N}_+$) are called* **extensional atoms**. *The atom $p(U^j) = p(U)\theta_j$ ($j \in \mathbb{N}_0$) is called the* **intensional atom** *in G_j, where $p(U^0) = p(X)$. $reduced_string(m) = \{A_i^j | 1 \le i \le n, 1 \le j \le m\}$ ($m \in \mathbb{N}_0 \cup \{\infty\}$) denotes the multiset of the extensional atoms within G_m, where $G_\infty = \cup_{i=1}^{\infty} G_i$.*

Definition 2.3 *Consider an SLF sirup $LP = \{r, e_1, \ldots, e_h\}$ given by*

$$\begin{array}{lrcl} r: & p(X) & :- & p(U), a_1(U_1), \ldots, a_n(U_n). \\ e_i: & p(X) & :- & b_i(X). \end{array}$$

(i) *The* **m-th iterate** *of r is given by r^m ($m \in \mathbb{N}_+$),*
the **compiled exit rules** *are given by e_i^m ($1 \leq i \leq h$, $m \in \mathbb{N}_0$):*

$$\begin{aligned} r^m: \ & p(X) \ :- \ p(U^m), reduced_string(m). \\ e_i^m: \ & p(X) \ :- \ b_i(U^m), reduced_string(m). \end{aligned}$$

(ii) *The* **m-th iterate** *of LP is given by the linear sirup LP^m ($m \in \mathbb{N}_+$),*
the **compiled version** *of LP is given by the logic program LP^ω:*

$$\begin{aligned} LP^m &= \{r^m, e_i^j | 1 \leq i \leq h, 0 \leq j \leq m-1\}, \\ LP^\omega &= \{ \ e_i^m \mid 1 \leq i \leq h, m \in \mathbb{N}_0 \ \}. \end{aligned}$$

An SLF sirup LP can be replaced by each of its iterates or by the infinite, non-recursive logic program LP^ω, since they are **p-equivalent** to LP, where p is the recursive predicate symbol in LP, cf. [Sei 89].

The concept of **k-sided linear sirups** introduced in [Na 87] refers to the sharing of arguments between the different atoms in the sets $reduced_string(m)$, i.e. to the connected components within the **hypergraph** that has one hyperedge $\{v_1, \ldots, v_h\}$ for each argument vector $(v_1, \ldots, v_h)$ that occurs in $reduced_string(m)$:

Definition 2.4

(i) *Given a goal $G = p_1(X_1), \ldots, p_h(X_h)$.*
An atom $p_{i_1}(X_{i_1})$ is **connected** *to an atom $p_{i_2}(X_{i_2})$ within G, iff $p_{i_1}(X_{i_1})$ shares an argument with $p_{i_2}(X_{i_2})$ or with an atom $p_{i_3}(X_{i_3})$ that is connected to $p_{i_2}(X_{i_2})$.*
A **connected component** *within G is a maximal connected subset of G.*

(ii) *Given a linear datalog sirup LP with the property (SLF-2).*
LP is **k-sided** *($k \in \mathbb{N}_0$), iff there exists some $c \in \mathbb{N}$, such that for any $c' > c$*
- *some $reduced_string(m)$ has k connected components C of size $|C| \geq c'$, and*
- *no $reduced_string(m)$ has more than k connected components C of size $|C| \geq c'$.*

Example 2.5 (continued) *The* **iterate** r^3 *of r is given by*

$$r^3: \ p(x_1, x_2, x_3, x_4, x_5, x_6) \ :- \ p(u_1^3, u_2^3, u_3^3, u_3^2, x_5, u_6^3), reduced_string(3).$$

where $reduced_string(3)$ consists of the following 7 **connected components**:

$C_1 = \{a_1(x_1, u_1^1), a_1(u_1^1, u_1^2), a_1(u_1^2, u_1^3)\}$,
$C_2 = \{a_2(x_2, u_3^1), a_3(u_3^1, u_2^3)\}$, $C_3 = \{a_3(x_3, u_2^2), a_2(u_2^2, u_3^3)\}$, $C_4 = \{a_3(x_4, u_2^1), a_2(u_2^1, u_2^2)\}$,
$C_{5,1} = \{a_4(x_6)\}$, $C_{5,2} = \{a_4(u_6^1)\}$, $C_{5,3} = \{a_4(u_6^2)\}$.

The compiled version LP^ω, i.e. the compiled exit rules e_i^m, can be evaluated by a **wave-front technique**, where independent **frontier relations** are assigned to the connected components of $reduced_string(m)$ ($m \in \mathbb{N}_+$). The frontier relations obtained for e_i^m can be used for the evaluation of $e_i^{m'}$ ($m' \geq m$), since the strings and the connected components are growing with m. The sidedness of a sirup is the number of infinitely growing connected components.

E.g. the n-dimensional linear datalog sirup LP in linear variable pattern of section 1.2 is n-sided, and the frontier relations could be identified easily. But there is **no simple rule** for dealing with the frontier relations, if LP is in no special normal form.

2.2 The Augmented Substitution Graph

The basic concept for the analysis of the expansion of an SLF sirup is the so-called **augmented substitution graph (AS-graph)**, which is a directed, binary, edge-labeled graph (with parallel edges and self-loops). Its definition refers to definition 2.1. By considering special paths in the AS-graph, the following **questions** will be answered:

(1) Which **intensional atoms** $p(U^j)$ share arguments ?

(2) Which **extensional atoms** A_i^j share arguments
or are connected within $reduced_string(m)$?

An **edge** e is described by a pair (v, v') of nodes and its length $l(e) \in \mathbb{R}$.
e can also be denoted by $(v \xrightarrow{l} v')$ and is considered equal to the edge $(v' \xrightarrow{-l} v)$.

A **path** π from a node v_0 to a node v_m is given by a sequence $\pi = (e_1, \ldots, e_m)$ ($m \in \mathbb{N}_+$) of edges $e_i = (v_{i-1} \xrightarrow{l_i} v_i)$ ($1 \leq i \leq m$). The length of π is $l(\pi) = \sum_{i=1}^{m-1} l(e_i)$, the start and end node of π is $\alpha(\pi) = v_0$ and $\omega(\pi) = v_m$, respectively. π is a **cycle**, iff $v_0 = v_m$. The **type** of π is given by $(v_0 \xrightarrow{l(\pi)} v_m)$. The $j-th$ **subpath** ($0 \leq j \leq m$) of π is given by $\pi_{1..j} = (e_1, ..., e_j)$, where $\pi_{1..0} = ()$ is an empty sequence of the length $l(\pi_{1..0}) = 0$.

The **substitution graphs (S-graphs)** introduced in [JaAgNe 87] do not completely describe the sharing of arguments between the intensional atoms $p(U^j)$, cf. [Sei 89].
We will define so-called **S^*-graphs**, which contain some additional edges to model this sharing of arguments completely.

Definition 2.6

(i) *The* **S-graph** $G_0 = (V_0, E_0)$ *is given by:*
- $V_0 := < 1, n' >$, *where* $< 1, m > := \{1, 2, \ldots, m\}$, *for all* $m \in \mathbb{N}_0$,
- $E_0 := \{ (i \xrightarrow{1} j) \mid x_i = u_j, 1 \leq i, j \leq n' \}$.

(ii) *The* **S^*-graph** $G_0^* = (V_0^*, E_0^*)$ *is given by:*
- $V_0^* := V_0 \cup \{ \eta \subseteq < 1, n' > \ maximal \mid (\ |\eta| \geq 2 \ ) \ and \ (\ \forall i, j \in \eta : \ u_i = u_j \ ) \}$,
- $E_0^* := E_0 \cup \{ (\eta \xrightarrow{1} i) \mid \eta \in V_0^* \setminus V_0, i \in \eta \}$.

Substitution graphs for linear datalog sirups LP with the property (SLF-2) are analysed in [JaAgNe 87]:

- Every node in the S-graph has at most one predecessor. Thus, the S-graph consists of trees and disjoint cycles, with trees rooted at nodes of the cycles. Every cycle is uni-directional, i.e. its edges e either all have length $l(e) \geq 0$ or all have length $l(e) \leq 0$. Different cycles in the S-graph never are connected by a path.
- If LP is in **CFTC**, then the S-graph contains no trees and no forward cycles (i.e. cycles whose edges e all have length $l(e) \geq 0$). There exists a CFTC sirup LP' with the property (SLF-2) that is p-equivalent to LP.

Question (1) is answered by the following theorem concerning the S^*-graph.

Theorem 2.7
Given an SLF sirup. The positions i_1 in U^{j_1} and i_2 in U^{j_2} are equal,
iff there is a path π of the type $(i_1 \stackrel{j_2-j_1}{\longrightarrow} i_2)$ in the S^*-graph, such that $j_1 + l_{min}(\pi) \geq 0$,
where $l_{min}(\pi) \;=\; min\{\, l(\pi_{1..j}) \mid 0 \leq j \leq m \,\}$.

We define the **augmented substitution graph (AS-graph)**, which models the 'interaction' of the intensional atoms $p(U^j)$ and the extensional atoms A_i^j.

Definition 2.8
The **AS-graph** $G_1 = (V_1, E_1)$ *is given by*

- $V_1 \;:=\; V_0^* \;\cup\; \{\, a_1, \ldots, a_n \,\}$,
- $E_1 \;:=\; E_0^* \;\cup\; \{\, (a_i \stackrel{1/2}{\rightarrow} j) \mid u_j \text{ occurs in } U_i \,\} \;\cup\; \{\, (j \stackrel{1/2}{\rightarrow} a_i) \mid x_j \text{ occurs in } U_i \,\}$.

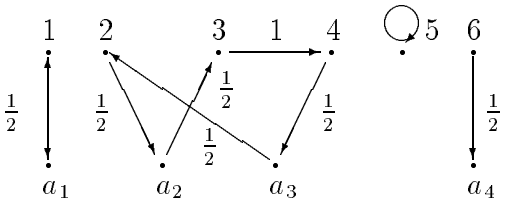

Figure 1: AS-graph of example 2.2

The answer to **question (2)** refers to paths in the AS-graph that have certain properties concerning the lengths of all subpaths and of those subpaths that end at an extensional predicate symbol.

Theorem 2.9
Given an SLF sirup and two different atoms $A_{i_1}^{j_1}, A_{i_2}^{j_2}$ in some strings of its expansion.

(i) $A_{i_1}^{j_1}, A_{i_2}^{j_2}$ *share a common argument,*
iff there is a path $\pi = (e_1, \ldots, e_l)$ of the type $(a_{i_1} \stackrel{j_2-j_1}{\longrightarrow} a_{i_2})$ in the AS-graph, such that
- *$l \geq 2$ and $\pi' = (e_2, \ldots, e_{l-1})$ either is a path in the S^*-graph or $\pi' = ()$, and*
- *$j_1 + l_{min}(\pi) \geq 1/2$.*

(ii) $A_{i_1}^{j_1}, A_{i_2}^{j_2}$ *are connected within $reduced_string(m)$ ($m \in I\!N_+ \cup \{\infty\}$),*
iff there is a path π of type $(a_{i_1} \stackrel{j_2-j_1}{\longrightarrow} a_{i_2})$ in the AS-graph, such that
- *$j_1 + l_{min}(\pi) \geq 1/2$ and*
- *$j_1 + a_{max}(\pi) \leq m$ (for $m = \infty$ this poses no restriction on π),*
where $a_{max}(\pi) \;=\; max\{\, l(\pi_{1..j}) \mid \omega(\pi_{1..j}) \in \{a_1, \ldots, a_n\}, 0 \leq j \leq m \,\}$.

The proof, which is given in [Sei 89], is based on the fact, that two different extensional atoms $A_{i_1}^{j_1}$ and $A_{i_2}^{j_2}$ share a **common argument** v, iff they share v with two intensional atoms $p(U^{k_1})$ and $p(U^{k_2})$ for some $k_1 \in \{j_1, j_1 - 1\}$ and $k_2 \in \{j_2, j_2 - 1\}$, respectively.

2.3 The Sidedness of SLF Sirups

Definition 2.10

(i) *A predicate symbol* a_i *(* $i \in <1,n>$ *) is* **recurrent**, *iff there exists a cycle* π *of the type* $(a_i \xrightarrow{l} a_i)$ *in the AS-graph, where* $l \neq 0$*; it is* **isolated**, *otherwise.*
$R = \{\ i \in <1,n> \mid a_i \text{ is recurrent } \}$, $S = <1,n> \setminus R$.

(ii) *The* **period** $m_{i,i}$ *of* a_i *(* $i \in R$ *) is given by the greatest common divisor of the lengths* $l(\pi)$ *of all cycles* π *of the type* $(a_i \xrightarrow{l(\pi)} a_i)$ *in the AS-graph, where* $l(\pi) > 0$.

Recurrent predicate symbols that are connected in the AS-graph have the same period, cf. [Sei 89].

Definition 2.11

(i) *Indices* $i, j \in R$ *are* **equivalent** *(* $i \sim j$ *), iff* $i = j$ *or there is a path from* a_i *to* a_j *in the AS-graph. The* **recurrent equivalence classes** *w.r.t.* $\sim$ *are denoted by* $R_1, \ldots, R_k$ *(i.e.* $R = \cup_{\nu=1}^{k} R_\nu$ *). The* **isolated equivalence class** *is formed by* $R_0 = S$.

(ii) *The* **period** l_ν *of* R_ν *(* $1 \leq \nu \leq k$ *) is the (common) period of all* a_i *(* $i \in R_\nu$ *).*
$K := \sum_{\nu=1}^{k} l_\nu$, $L := lcm\{\ l_\nu \mid 1 \leq \nu \leq k\ \}$.

K and L are **characteristics** of an SLF sirup LP. The following **main theorem** says that LP is K-sided, L will be referred to later.

Theorem 2.12 (Sidedness of SLF sirups)
An SLF sirup LP *is* K*-sided, where* $K = \sum_{\nu=1}^{k} l_\nu$. *Every iterate* LP^m *(* $m \in \mathbb{N}_+$ *) is* K*-sided, too.*

Sketch of the proof: cf. [Sei 89]

For $i, j \in R_\nu$ ($1 \leq \nu \leq k$), such that a_i and a_j are connected in the AS-graph, there exists a (unique) minimal, non-negative distance $m_{i,j}$ ($0 \leq m_{i,j} \leq l_\nu - 1$) from a_i to a_j in the AS-graph.
For $i, j \in S$, there exists a unique distance $m_{i,j}$ from a_i to a connected a_j.

The following **equivalence relation** on $reduced_string(\infty)$ is a useful coarsening of the connected components:

$A_{i_1}^{j_1}, A_{i_2}^{j_2} \in reduced_string(\infty)$ are called **equivalent** ($A_{i_1}^{j_1} \equiv A_{i_2}^{j_2}$), iff
($i_1, i_2 \in R_\nu$ and $j_2 \equiv j_1 + m_{i_1,i_2} \pmod{l_\nu}$) or ($i_1, i_2 \in S$).

We fix an arbitrary set $\wp$ of paths in the AS-graph that contains one path $\pi_{i,j}$ of type $(a_i \xrightarrow{m_{i,j}} a_j)$ for every $i, j \in R_\nu$ ($1 \leq \nu \leq k$) and one path $\pi_{i,j}$ of type $(a_i \xrightarrow{m_{i,j}} a_j)$ for every $i, j \in S$, where a_i and a_j are connected in the AS-graph.
We define $\tau_1 = max\{a_{max}(\pi) | \pi \in \wp\}$, $\tau_2 = max\{-l_{min}(\pi) | \pi \in \wp\} - 1/2$.

For all $m \geq \tau_1 + \tau_2$, the following can be shown:

- Given $j_1, j_2 \in < \tau_2 + 1, m - \tau_1 >$:
  for $i_1 \in R$, it holds: $A_{i_1}^{j_1}$ and $A_{i_2}^{j_2}$ are connected in $reduced_string(m)$, iff $A_{i_2}^{j_2} \equiv A_{i_1}^{j_1}$;
  for $i_1 \in S$, it holds: if $A_{i_1}^{j_1}$ is connected to $A_{i_2}^{j_2}$ in $reduced_string(m)$, then $i_2 \in S$.

- The complement $< 1, \tau_2 > \cup < m - \tau_1 + 1, m >$ contains $\tau_2 + \tau_1$ elements, for all m.

Thus, the connected components containing atoms with recurrent predicate symbols are growing infinitely, whereas the size of the other connected components is at most $|S|$.

Hence, LP is K-sided.

Finally, it is easy to show that LP^m ($m \in I\!N_+$) is K-sided, too.

2.4 Determination of the Sidedness of SLF Sirups

The folding operator:

Consider a binary, edge-weighted graph $G = (V, E)$:
A pair $b = (e_1, e_2)$ of edges $e_i = (v \xrightarrow{l_i} v_i)$, where $0 < l_1 \leq l_2$, is called a **branch**.
The **folding** of G w.r.t. a branch b is defined by $fold(\ G, b\ ) := (V', E')$:

(i) if $l_1 = l_2$, then the string $v_1 v_2$ denotes a new node and

- $V' = (\ V \setminus \{v_1, v_2\}\ ) \cup \{v_1 v_2\}$,
- $E' = (\ E \setminus \{\ (v_1 \xrightarrow{l} v'), (v_2 \xrightarrow{l} v') \in E \mid l \in I\!R, v' \in V\ \}\ ) \cup$
  $\{\ (v_1 v_2 \xrightarrow{l} v') \mid (v_1 \xrightarrow{l} v') \in E \text{ or } (v_2 \xrightarrow{l} v') \in E,\ l \in I\!R, v' \in V\ \}$.

(ii) if $l_1 \neq l_2$, then $V' = V$ and $E' = (\ E \setminus \{\ (v \xrightarrow{l_2} v_2)\ \}\ ) \cup \{\ (v_1 \xrightarrow{l_2 - l_1} v_2)\ \}$.

The intention of the folding operator is to **simplify the structure** of the graph G while **preserving the path lengths**:

- A path π of the type $(v_1 \xrightarrow{l} v_2)$ in G corresponds to a path π' of the type $(v_1' \xrightarrow{l} v_2')$ in G', where v_i' ($i = 1, 2$) is the unique node in V' that contains the substring v_i.

- Assume that all edges have a length $l \in \{1/2, 1\}$. Then the **potential** of G defined by $pot(G) = |V| + |\{(v \xrightarrow{1} v') \in E\}|$ has the property that $pot(fold(G, b)) < pot(G)$ and $pot(G) \in I\!N_0$. Thus, there are at most $pot(G)$ successive foldings applicable to G.

The folding algorithm:

The following algorithm **FOLD** transforms the AS-graph (V_1, E_1) to a **folded AS-graph** (V_1'', E_1'') by a sequence of successive foldings. The sidedness of LP can be derived from (V_1'', E_1'').

It holds $|V_1| \leq n + 1.5n'$, $|E_0^*| \leq 2n'$ and thus, there are at most $pot(\ (V_1, E_1)\ ) = |V_1| + |E_0^*| \leq n + 3.5n'$ foldings operations, where each can be performed in $n + n'$ elementary operations.

algorithm FOLD$((V_1, E_1), (V_1'', E_1''))$;
$(V_1', E_1') := (V_1, E_1)$;
while (there is a **branch** b) **do**
$\quad (V_1', E_1') := fold(\ (V_1', E_1'), b\ )$;
$V_1'' := \{\ \{v_1, \ldots, v_m\} | v_1 \ldots v_m \in V_1'\ \}$;
$E_1'' := \{\ (\{v_1, \ldots, v_m\} \xrightarrow{l} \{w_1, \ldots, w_{m'}\}) | (v_1 \ldots v_m \xrightarrow{l} w_1 \ldots w_{m'}) \in E_1'\ \}$.

Theorem 2.13 *cf. [Sei 89]*
Consider an SLF-sirup LP and its folded AS-graph (V_1'', E_1'').

(i) *Function FOLD has the Church-Rosser property (* (V_1'', E_1'') *is independent of the sequence of foldings), and its complexity is in* $O(max\{n^2, n'^2\})$.

(ii) *All paths and cycles in* (V_1'', E_1'') *are* **uni-directional***; their nodes are pairwise not connected by a path.* V_1'' *is a disjoint partition of* V_1.

(iii) *Let* $a_i'' \in V_1''$ *contain* $a_i \in V_1$ *(* $1 \leq i \leq n$ *). Let* $\pi_1, \ldots, \pi_k$ *be those cycles in* (V_1'', E_1'') *that traverse nodes* $a_i'' \in V_1''$ *(* $i \in < 1, n >$ *), let* l_ν *be the* **length** *of* π_ν, $W_\nu \subseteq V_1''$ *the set of nodes traversed by* π_ν *and* $R_\nu = \{i \in < 1, n > | a_i'' \in W_\nu\} \subseteq < 1, n >$ *(* $1 \leq \nu \leq k$ *).*
Then LP is K-sided, where $K := \sum_{\nu=1}^{k} l_\nu$, *with the recurrent equivalence classes* R_ν, *whose periods are* l_ν.

Example 2.14 (continued) *The folded AS-graph shows that LP is 4-sided with*

- *the* **recurrent** *equivalence classes* $R_1 = \{1\}$, $R_2 = \{2, 3\}$,
  the **isolated** *equivalence class* $R_0 = \{4\}$,
- *the* **periods** $l_1 = 1$, $l_2 = 3$, *i.e.* $K = 4$, $L = 3$.

3 Decomposition of k-Sided SLF Sirups

The **canonical** k**-sided normal form (CKNF)** is a special normal form for SLF sirups. It is introduced in [Sei 89]. Every k-sided SLF sirup can be normalized to an equivalent k-sided CKNF sirup.

Every k-sided CKNF sirup can be **decomposed** by **counting techniques** into k lower-dimensional GFTC sirups, which can be evaluated very efficiently.

3.1 The Canonical k-Sided Normal Form

The **decomposition** of SLF sirups in linear variable pattern by the **counting method** was based on the fact that there was one infinite connected component $\{A_i^j | j \in I\!N_+\}$ in

$reduced_string(\infty)$ for each recurrent predicate symbol a_i ($1 \leq i \leq n$).

For CFTC sirups, this means that all recurrent equivalence classes are singletons with the period 1. Such sirups are in **canonical k-sided normal form (CKNF)**, cf. [Sei 89]:

Definition 3.1
An SLF sirup $LP = \{r, e_1, \ldots, e_h\}$ *is in* **canonical** k**-sided normal form (CKNF)**, *iff the argument vectors* $X_i, Y_i^0, Z_i^0, Y_i, Z_i, V_i, X_S, U_S, W$ *in the linear recursive rule* r *are disjoint, where*

$$\begin{aligned} r:\ & p(\ldots, X_i, Y_i^0, Z_i^0, \ldots, X_S, W) \ :-\ p(\ldots, X_i, Y_i, Z_i, \ldots, U_S, W), \\ & a_1(X_1, Y_1^0, Y_1, Z_1^0, V_1), \ldots, a_k(X_k, Y_k^0, Y_k, Z_k^0, V_k), \\ & a_{k+1}(U_{k+1}), \ldots, a_n(U_n). \end{aligned}$$

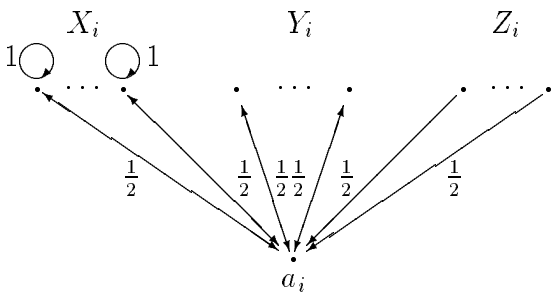

Figure 2: argument positions for CKNF connected to a recurrent a_i

Every SLF sirup LP can be **normalized** to an equivalent CKNF sirup LP' with the same sidedness and dimension. LP' can be derived from the iterate LP^L of LP, where $L = lcm\{l_\nu | 1 \leq \nu \leq k\}$, cf. [Sei 89]. This shows that the sidedness K is bounded by the dimension n' of LP.

Lemma 3.2
Consider a K*-sided,* n'*-dimensional SLF sirup* LP *and a predicate symbol* q.
Then it holds $0 \leq K \leq n'$, *and there is a* K*-sided,* n'*-dimensional CKNF sirup* LP' *that is* q*-equivalent to* LP.

Example 3.3 (continued) *The* **normalized** *CKNF sirup* $LP' = \{r', e_1, \ldots, e_5\}$ *for* LP *is derived from the connected components in* r^3 *(since* $L = 3$ *):*

$$\begin{aligned} r':\ & p(x_1, x_2, x_3, x_4, x_5, x_6) \ :-\ p(u_1^3, u_2^3, u_3^3, u_3^2, x_5, u_6^3), \\ & \qquad\qquad b_1(x_1, u_1^3), b_2(x_2, u_2^3), b_3(x_3, u_3^3), b_4(x_4, u_3^2), b_5(x_6). \end{aligned}$$

The exit rule e_i *(* $1 \leq i \leq 5$ *) has the* b_i*-atom in* r' *as its head and the atoms in the set* C_i *as its body, where* $C_5 = C_{5,1} \cup C_{5,2} \cup C_{5,3}$, *cf. example 2.2.*
LP' *is 4-sided just like* LP *and has*

- *the* **recurrent** *equivalence classes* $R_i' = \{i\}$ *with the* **periods** $l_i = 1$ *(* $1 \leq i \leq 4$ *),*
- *the* **isolated** *equivalence class* $R_0' = \{5\}$.

3.2 Decomposition of CKNF Sirups by Counting Techniques

Consider a k-sided CKNF sirup $LP = \{r, e\}$ according to the definitions 2.1 and 3.1. The **iterates** r^m ($m \in \mathbb{N}_+$) are given by

$$r^m: \; p(\ldots, X_i, Y_i^0, Z_i^0, \ldots, X_S, W) \; :- \; p(\ldots, X_i, Y_i^m, Z_i^m, \ldots, U_S^m, W),$$
$$\ldots, a_i(X_i, Y_i^0, Y_i^1, Z_i^0, V_i^1), \ldots, a_i(X_i, Y_i^{m-1}, Y_i^m, Z_i^{m-1}, V_i^m), \ldots .$$

(i) The recurrent predicate symbols:

Every recurrent predicate symbol a_i ($1 \leq i \leq k$) 'determines' those argument positions that are connected to a_i in the AS-graph. Figure 3 gives the lengths $l(v)$ of the corresponding argument vectors v:

v	X_i	Y_i^0, Y_i	Z_i^0, Z_i
l(v)	α_i	β_i	γ_i

Figure 3: lengths of argument vectors

The logic program LP_i defines a p_i-database:

- if $\beta_i \neq 0$, then $LP_i = \{r_i, e_i, s_i, t_i\}$ is a GFTC sirup derived from r^2:

$$\begin{aligned}
r_i: \; q_i(X_i, Y_i^0, Y_i^2, m+1) \; &:- \; q_i(X_i, Y_i^0, Y_i^1, m), a_i(X_i, Y_i^1, Y_i^2, Z_i^1, V_i^2). \\
e_i: \; q_i(X_i, Y_i^0, Y_i^2, 1) \; &:- \; a_i(X_i, Y_i^0, Y_i^2, Z_i^0, V_i^2). \\
s_i: \; p_i(X_i, Y_i^0, Y_i^2, Z_i^0, m+1) \; &:- \; a_i(X_i, Y_i^0, Y_i^1, Z_i^0, V_i^1), q_i(X_i, Y_i^1, Y_i^2, m). \\
t_i: \; p_i(X_i, Y_i^0, Y_i^2, Z_i^0, 1) \; &:- \; a_i(X_i, Y_i^0, Y_i^2, Z_i^0, V_i^2).
\end{aligned}$$

  if $\gamma_i = 0$, then LP_i can be simplified to the GFTC sirup $LP_i = \{r_i, e_i\}$ by replacing q_i in r_i and e_i by p_i.

- if $\beta_i = 0$, then $LP_i = \{e_i\}$ is a non-recursive logic program:

$$e_i: \; p_i(X_i, Z_i^0) \; :- \; a_i(X_i, Z_i^0, V_i^2).$$

For $\beta_i \neq 0$, the recursive, non-datalog rule r_i is called a **counting rule** and the last, numerical argument in its head atom is called the **generation counter**.
When comparing the **dimension** n_i' of the GFTC sirup LP_i with the number h_i' of argument positions that the literal $a_i(U_i)$ shares with the intensional atoms $p(X)$ and $p(U)$ in the rule r, where

$$\begin{aligned}
n_i' &= l(X_i) + l(Y_i^0) + l(Y_i^1) + 1 = \alpha_i + 2 \cdot \beta_i + 1, \\
h_i' &= l(X_i) + l(Y_i^0) + l(Z_i^0) = \alpha_i + \beta_i + \gamma_i,
\end{aligned}$$

it turns out that $n_i' \leq 2 \cdot h_i' + 1$ in general. For large k, the dimensions n_i' are much smaller than n'. This allows for the efficient **precomputation** and **storage** of all the p_i-databases, even for very large p-databases.

(ii) The isolated predicate symbols:

The X_S- and U_S^m-values for the iterates r^m can be produced by the non-recursive rules r_S^m, where $C_m = \{ A_i^j \mid i \in S, 1 \leq j \leq m \}$:

$$r_S^m : \quad p_0(X_S, U_S^m) \ :- \ C_m.$$

By theorems 2.7 and 2.9 for $m \geq s+2$, where s is the maximal distance in the AS-graph between two elements of S, no connected component in C_m shares arguments with both $p(X)$ and $p(U^m)$, and different C_m can be unified. Hence, all rules r_S^m ($m \geq s+2$) are equivalent.

A nice property of a logic program is the **boundedness** (**uniform boundedness**), cf. [NaSa 87], which means that there is some $m \in \mathbb{N}$, such that the bottom-up evaluation terminates after at most m iterations for all finite extensional databases (for all finite databases, respectively).

We define the non-recursive logic program $LP_0 = \{r_0\}$, where $r_0 = r_S^{s+2}$, which derives a p_0-database from the a_i-databases for the isolated, extensional predicate symbols. LP_0 is **uniformly bounded** and could be optimized by removing redundant parts according to the methods of [Na 89].

(iii) The 'combination' of the p_i-databases:

The logic program $LP_{k+1} = \{ r_{k+1}, e^i \mid 0 \leq i \leq s+1 \}$ identifies the generation counters via the non-recursive non-datalog rule r_{k+1}, where $T_i = (X_i, Y_i^0, Y_i, Z_i^0, m)$ for $\beta_i \neq 0$ and $T_i = (X_i, Z_i^0)$ for $\beta_i = 0$:

$$r_{k+1} : \quad p(X) \ :- \ a_0(U), p_0(X_S, U_S), p_1(T_1), \ldots, p_k(T_k), m \geq s+2.$$

Finally, the **canonical decomposition** of LP is given by $dec(LP) = \cup_{i=0}^{k+1} LP_i$:

Theorem 3.4 *cf. [Sei 89]*
The **canonical decomposition** *$dec(LP)$ of a CKNF sirup LP is p-equivalent to LP. $dec(LP)$ is* **uniformly bounded**, *if $k = 0$.*

Again for **cyclic** a_i-databases the **improved counting algorithm** of [HaNa 88], which was introduced for two-sided CKNF sirups in linear variable pattern, might be generalizable to the canonical decomposition of arbitrary k-sided CKNF sirups.

Example 3.5 (continued) *We decompose the CKNF sirup $LP'' = \{r'', e_1, \ldots, e_5\}$, which is a notationally simplified p-equivalent of LP and LP':*

$$r'' : \ p(x_1, x_2, x_3, x_4, x_5, x_6) \ :- \ p(u_1, u_2, u_3, u_4, x_5, u_6), \\ b_1(x_1, u_1), b_2(x_2, u_2), b_3(x_3, u_3), b_4(x_4, u_4), b_5(x_6).$$

$dec(LP'')$ is derived from $(r'')^2$, where $\beta_i \neq 0$, $\alpha_i = \gamma_i = 0$ ($1 \leq i \leq 4$), $s = 0$:

$$r_i'' : \ p_i(x_i, u_i^2, m+1) \ :- \ p_i(x_i, u_i^1, m), b_i(u_i^1, u_i^2).$$

$$
\begin{aligned}
e_i'' :\ p_i(x_i, u_i^2, 1) \quad &:- \quad b_i(x_i, u_i^2). \\
r_0'' :\ p_0(x_6) \quad &:- \quad b_5(x_6), b_5(u_6). \\
r_5'' :\ p(x_1, x_2, x_3, x_4, x_5, x_6) \quad &:- \quad a_0(u_1, u_2, u_3, u_4, x_5, u_6), \\
& \qquad p_0(x_6), p_1(x_1, u_1, m), \ldots, p_4(x_4, u_4, m), m \geq 2.
\end{aligned}
$$

Summarizing, the CKNF normalization of the 6-dimensional SLF sirup LP of example 2.2 has been decomposed into four 3-dimensional GFTC sirups and some non-recursive rules.

4 Final Remarks

The decomposition approach for CKNF sirups equivalently can be formulated as a **wave-front technique**, cf. [Sei 89], based on so-called composition operators, cf. [SiSo 88]. These associative operators induce powers of the extensional a_i-databases that can be computed by a logarithmic number of database operations. Further efficiency stems from the reduction of the dimension.

The expansion of an SLF sirup w.r.t. an arbitrary goal $G_0 = p(X')$ including constant symbols or repeated variables (instead of the goal $p(X)$ formed by the head of its recursive rule) corresponds to a logic program which poses a **selection** to the derived p-database. It should be possible to get optimized evaluation plans for this program, too.

An interesting problem would be to **generalize the AS-graphs** and the results from SLF sirups to linear non-datalog sirups or even to arbitrary logic programs.

References

[AhUl 79] **A. Aho, J. Ullman:** Universality of Data Retrieval Languages, Proc. POPL 1979.

[AgDe 88] **R. Agrawal, P. Devanbu:** Moving Selections into Linear Least Fixpoint Queries, Proc. Intl. Conf. on Data Engineering, 1988.

[BaMaSaUl 86] **F. Bancilhon, D. Maier, Y. Sagiv, J. Ullman:** Magic Sets and Other Strange Ways to Implement Logic Programs, Proc. PODS 1986.

[BaRa 86] **F. Bancilhon, R. Ramakrishnan:** An Amateur's Introduction to Recursive Query Processing Strategies, Proc. ACM SIGMOD 1986.

[BeeRa 87] **C. Beeri, R. Ramakrishnan:** On the Power of Magic, Poc. PODS 1987.

[BiSt 88] **J. Biskup, H. Stiefeling:** Transitive Closure Algorithms for Very Large Databases, Proc. Intl. Workshop on Graph-Theoretic Concepts in Computer Science 1988 (WG'88), LNCS 344, Springer, 1989.

[GüKiBa 87] **U. Güntzer, W. Kiessling, R. Bayer:** On the Evaluation of Recursion in (Deductive) Database Systems by Efficient Differential Fixpoint Iteration, Proc. Intl. Conf. on Data Engineering 1987.

[HaHe 87] **J. Han, L. Henschen:** Handling Redundancy in the Processing of Recursive Database Queries, Proc. ACM SIGMOD 1987.

[HaLu 86] **J. Han, H. Lu:** Some Performance Results on Recursive Query Processing in Relational Database Systems, Proc. Intl. Conf. on Data Engineering 1986.

[HaNa 88] **R. Haddad, J. Naughton:** Counting Methods for Cyclic Relations, Proc. PODS 1988.

[HeNa 84] **L. Henschen, S. Naqvi:** On Compiling Queries in Recursive First-Order Data Bases, JACM, vol. 31(1), 1984.

[Io 85] **Y. Ioannidis:** A Time Bound on the Materialization of Some Recursively Defined Views, Proc. VLDB 1985.

[IoWo 86] **Y. Ioannidis, E. Wong:** An Algebraic Approach to Recursive Inference, in: **L. Kerschberg:** Expert Database Systems, Proc. First Intl. Workshop 1986, Benjamin-Cummings, Menlo Park, CA, 1987.

[JaAgNe 87] **H. Jagadish, R. Agrawal, L. Ness:** A Study of Transitive Closure as a Recursion Mechanism, Proc. ACM SIGMOD 1987.

[KiLo 86] **M. Kifer, E. Lozinskii:** Filtering Data Flow in Deductive Databases, Proc. ICDT 1986.

[Lo 84] **J. Lloyd:** Foundations of Logic Programming, Springer, 1984.

[Na 87] **J. Naughton:** One-Sided Recursions, Proc. PODS 1987.

[Na 89] **J. Naughton:** Minimizing Function-Free Recursive Inference Rules, JACM, vol. 36(1), 1989.

[NaSa 87] **J. Naughton, Y. Sagiv:** A Decidable Class of Bounded Recursions, Proc. PODS 1987.

[Sa 87] **Y. Sagiv:** Optimizing Datalog Programs, Proc. PODS 1987.

[SaZa 87] **D. Sacca, C. Zaniolo:** Magic Counting Methods, Proc. ACM SIGMOD 1987.

[Sei 88] **D. Seipel:** A Graph-Based Decomposition Approach for Recursive Query Processing, Proc. Intl. Workshop on Graph-Theoretic Concepts in Computer Science 1988 (WG'88), LNCS 344, Springer, 1989.

[Sei 89] **D. Seipel:** Decomposition in Database and Knowledge-Base Systems, Doctoral Dissertation, University of Würzburg, 1989.

[SiSo 88] **S. Sippu, E. Soisalon-Soininen:** A Generalized Transitive Closure for Relational Queries, Proc. PODS 1988. (and: Research Report 183, Univ. Karlsruhe, Dept. of CS, 1987).

[Ul 85] **J. Ullman:** Implementation of Logical Query Languages for Databases, ACM TODS, vol. 10(3), 1985.

[YoHeHa 88] **C. Youn, L. Henschen, J. Han:** Classification of Recursive Formulas in Deductive Databases, Proc. ACM SIGMOD 1988.