

Chapter 2

Parallel and Distributed Logic Programming

This chapter provides a new approach to reason with logic programs using a specialized data structure similar to Petri nets. Typical logic programs include a number of concurrently resolvable clauses. A priori detection of these clauses indeed is useful for their subsequent participation in the concurrent resolution process. The chapter explores the scope of distributed mapping of program clause components onto Petri nets so as to automatically select the participant clauses for concurrent resolution. An algorithm for concurrent resolution of clauses on Petri nets has been undertaken with a motive to improve the speed-up factor for the program without sacrificing the resource utilization rate. Examples have been introduced to illustrate all new concepts. The exercises at the end of the chapter include a number of interesting problems with sufficient hints to each so as to enable the readers to test their level of understanding.

2.1 Introduction

Logic programming has already earned much importance for its increasing applications in data and knowledge engineering. A logic program usually consists of a special type of program clauses known as **horn clauses**. Programs built with horn clauses only are called normal logic programs. Complex knowledge having multiple consequent literals cannot be represented by normal logic programs because of its structural restriction imposed by horn clauses.

In spite of limitations in knowledge representation, normal logic programs are still prevalent in relational languages like PROLOG and DATALOG for simplicity in designing their compilers. Generally, compilers for logic programming employ linear resolution that resolves each pair of program clauses at a time. Execution of a program by linear resolution thus requires a

considerable amount of time. Most logic programs usually include a number of concurrently resolvable clauses; unfortunately there is hardly any literature on parallel and distributed models of logic programming, capable of resolving multiple program clauses concurrently.

In the early 1990s, Patt [12] examined a nonconventional execution model of a uniprocessor micro-engine for a PROLOG program and measured its performance with a set of 14 benchmarks. Yan [15] provided a novel scheme for concurrent realization of PROLOG on a RAP-WAM machine.¹ The WAM generates an intermediate code during the compilation phase of a PROLOG program that exploits the fullest degree of parallelism in the program itself. The RAP, on the other hand, reduces the overhead associated with the run-time management of *variable binding conflict* between goals. Hermenegildo and Tick proposed an alternative model [3] for concurrent execution of PROLOG programs by RAP-WAM combinations by representing the dependency relationship of the program clauses by a graph, where the take-off arcs at the vertex in the graph denotes parallel tasks in the program. They employed goal stacks with each processing element. When a parallel call is invoked, the concurrent tasks are mapped autonomously to the stacks of the less busy or idle processing elements. A synchronization and co-ordination for parallel calls was also implemented in their scheme.

In recent times, Ganguly, Silberschatz, and Tsur [2] presented a new algorithm for automated mapping of logic programs onto a parallel processing architecture. In their first scheme, they considered the mapping of program clauses having shared variables onto adjacent processors. This reduces the communication overhead among the program clauses. In the latter part of their work, they eliminated the above constraints at the cost of extra network management time. Takeuchi [14] presented a new language for AND-OR parallelism. Kale [6] discussed the scope of an alternative formulation of problem-solving using parallel AND-OR trees. Among the existing speed-up schemes of logic programming machines, the content addressable memory (CAM)-based architecture of PROLOG machines by Naganuma et al. [11] needs special mention. To speed up the execution performance of PROLOG programs, they employed *hierarchical pipelining and garbage collection mechanism* of a CAM for efficient backtracking on a SLD tree.

Though a number of techniques are prevalent for the realization of logic programs on a parallel architecture, none of these are capable of representing the theoretically possible maximum parallelism in a program. For realization of all types of parallelism in a logic program, a specialized data structure appropriate for representing the possible parallelism is needed. Petri net has already proved

¹ RAP is an acronym for **R**estricted **A**ND-**P**arallelism, and WAM is an acronym for **W**arren **A**bstract **M**achine.

itself as a successful data structure for reasoning with complex rules. For instance, rules having more than one antecedent and consequent clause with each clause containing a number of variables can be represented by a Petri net structure [8]. Murata [10] proposed the scope of Petri net models for knowledge representation and reasoning under the framework of predicate logic.

A number of researchers are working on complex reasoning problems using Petri nets [5], [9], [13]. Because of the distributed structure of a Petri net, pieces of knowledge fragmented into components can easily be mapped onto this structure. For example, a transition firing in a Petri net may synonymously be used as rule firing in an expert system. The transitions may be regarded as the implication operator of a rule, while its input and output places may, respectively, be regarded as the antecedent and consequent clauses of a rule. The arguments associated with the predicates of a clause are also assigned at the arcs connecting the place containing the predicate and the transition describing the implication rule. Such fragmentation and mapping of the program components onto different modules of a Petri net enhances the scope of parallelism in a logic program. The objective of the chapter is to fragment a given logic program to smallest possible units, and map them onto a Petri net to fully exploit its parallelism.

The methodology of reasoning presented in the chapter is an extension of Murata's classical models on Petri nets [10], [13], applied to logic programming. Murata defined a set of rules to synonymously describe the resolution of horn clauses in a normal logic program with the firing of transitions in a Petri net. The chapter attempts to extend Murata's scheme for automated reasoning to non-Horn clause based programs as well.

Section 2.2 provides related definitions of important terminologies used in this chapter. The concept of concurrency in resolution process is introduced in Section 2.3. A new model for concurrent resolution in Petri nets is presented in Section 2.4. Performance analysis of the Petri net-based model is covered in Section 2.5. Conclusions are listed in Section 2.6. A set of numerical problems has been undertaken in the exercises of the chapter.

2.2 Formal Definitions

2.2.1 Preliminary Definitions

Definition 2.1: A clause cl_i is represented by

$$A_i \leftarrow B_i. \quad (2.1)$$

B_i denotes the body, A_i denotes the head, and ' \leftarrow ' denotes the implication operator. The body B_i usually is a conjunction of literals $B_{il} \exists j$, i.e.,

$$B_i = B_{i1} \wedge B_{i2} \wedge \dots \wedge B_{il} \quad (2.2)$$

The head A_i usually is a disjunction of literals $A_{ik} \exists k$, i.e.,

$$A_i = A_{i1} \vee A_{i2} \vee \dots \vee A_{ij} \quad (2.3)$$

The literals A_{ik} and B_{il} have arguments containing terms that may include variables (denoted by capital letters), constants (denoted by small letters), function of variables, and function of function of variables (in a recursive form).

Example 2.1: This example illustrates the constituents of a clause. For instance,

$$\text{Father}(Y, Z), \text{Uncle}(Y, Z) \leftarrow \text{Father}(X, Y), \text{Grandfather}(X, Z). \quad (2.4)$$

is an example of a general clause, where the body consists of $\text{Father}(X, Y)$ and $\text{Grandfather}(X, Z)$ and the head consists of $\text{Father}(Y, Z)$ and $\text{Uncle}(Y, Z)$. The clause states that if X is the father of Y and X is the grandfather of Z , then either Y is the father of Z or Y is the uncle of Z .

In case all the terms are bound variables or constants, the literals A_{ij} or B_{il} are called **ground literals**.

Example 2.2: The following is an example of a clause with all variables bound by constants, thereby resulting in ground literals $\text{Father}(n, a)$ and $\text{Son}(a, n)$.

$$\text{Father}(n, a) \leftarrow \text{Son}(a, n). \quad (2.5)$$

The above clause states that if a is son of n then n is the father of a .

Special cases:

- i) In case of a **goal clause (query)** the consequent part A_i is absent.

The clause (2.6) presented below contains no consequent part, and hence it is a query.

$$\leftarrow \text{Grandfather}(X, Z). \quad (2.6)$$

Given that X is the Grandfather of Z , the clause (2.6) questions the value of X and Z .

- ii) A clause with an empty body and consisting of ground literals in the head is regarded as a **fact**.

The clause (2.7) below contains no body part and the variable arguments are bound by constants. Thus, it is a fact.

$$\text{Grandfather}(r, a) \leftarrow. \quad (2.7)$$

It states that r is Grandfather of a .

- iii) When the consequent part A_i includes a single literal, the resulting clause is called a **horn clause**. The details about horn clauses are given in Definition 2.2.
- iv) When A_{ik} and B_{ij} do not include arguments, we call them propositions and the clause " $A_i \leftarrow B_i$." is then called a **propositional clause**.

The clause (2.8) below for instance is a propositional clause as it does not contain any arguments.

$$P \leftarrow Q, R. \quad (2.8)$$

Definition 2.2: A **horn clause** contains a head and a body with at most one literal in its head.

Example 2.3: The clause (2.9) is an example of a horn clause.

$$P \leftarrow Q_1, Q_2, \dots, Q_n. \quad (2.9)$$

It represents a horn clause where P and the Q_i are literals or atomic formulas. It means if all the Q_i s are true, then P is also true. Q_i is the body part and P is the head in this horn clause.

Definition 2.3: A clause containing more than one literal in its head is known as a **non-horn clause**.

Example 2.4: The clause (2.10) for instance is a non-horn clause.

$$P_1, P_2, \dots, P_m \leftarrow Q_1, Q_2, \dots, Q_n. \quad (2.10)$$

Definition 2.4: An **extended horn clause (EHC)** contains a head and a body with at least one clause in the body and zero or more clauses in its head.

Commas are used to denote conjunction of the literals in the body and disjunction of literals in the head.

Example 2.5: The general format of an EHC is

$$A_1, A_2, \dots, A_n \leftarrow B_1, B_2, \dots, B_m. \quad (2.11)$$

Here the head and the body contain n and m number of literals, respectively.

It is important to note that an extended horn clause includes both horn clause and its extension as well.

Definition 2.5: A program that contains extended horn clauses, as defined above, is called an **extended logic program**.

Example 2.6: The clauses (2.12–2.15) together represent an extended logic program. It includes extended horn clause (2.12) with facts (2.13–2.15).

$$\text{Father}(Y, Z), \text{Uncle}(Y, Z) \leftarrow \text{Father}(X, Y), \text{Grandfather}(X, Z). \quad (2.12)$$

$$\text{Father}(r, d) \leftarrow. \quad (2.13)$$

$$\neg \text{Father}(d, a) \leftarrow. \quad (2.14)$$

$$\text{Grandfather}(r, a) \leftarrow. \quad (2.15)$$

To represent the query “whether d is uncle of a ?” the goal clause of the following form may be constructed.

$$\text{Goal: } \leftarrow \text{Uncle}(d, a). \quad (2.16)$$

The answer to the query can be obtained by taking negation of the goal and then resolving it with the supplied clauses (2.12–2.15). In the present context, the answer to the query will be *true*.

To explain this, we need to introduce the principles of **resolvability of two clauses**. In order to explain resolvability of clauses we further need to introduce **substitution sets** and **most general unifier**.

Definition 2.6: A substitution represented by a set of ordered pairs $s\{t_1/v_1, t_2/v_2, \dots, t_n/v_n\}$ is called the **substitution set**. The pair t_i/v_i means that the term t_i is substituted for every occurrence of the variable v_i throughout.

Example 2.7: There exist four substitution sets for the predicate $P(a, Y, f(Z))$ in the following instances:

$P(a, X, f(W))$
 $P(a, Y, f(b))$
 $P(a, g(X), f(b))$
 $P(a, c, f(b))$

Substitution sets for the above examples are

$s_1 = \{X/Y, W/Z\}$
 $s_2 = \{b/Z\}$
 $s_3 = \{g(X)/Y, b/Z\}$
 $s_4 = \{c/Y, b/Z\}$

To denote a substitution instance of an expression w , using a substitution s we write ws .

Example 2.8: Let the expression $w = P(a, Y, f(Z))$, the substitution set $s = \{X/Y, W/Z\}$. Then the substitution instance $ws = P(a, X, f(W))$.

2.2.2 Properties of the Substitution Set

Property 1: $(ws_1)s_2 = w(s_1s_2)$ where w is an expression and s_1s_2 are two substitutions.

Example 2.9: To illustrate the above property, let

$w = P(X, Y),$
 $s_1 = \{f(Y)/X\}$
 and $s_2 = \{a/Y\}.$

Now, $(ws_1)s_2 = (P(f(Y), Y))\{a/Y\}$
 $= (P(f(a), a)).$

Again, $w(s_1s_2) = (P(X, Y))\{f(a)/X, a/Y\}$
 $= P(f(a), a).$

Therefore, $(ws_1)s_2 = w(s_1s_2).$

The composition of two substitutions s_1, s_2 is hereafter denoted by $s_1 \Delta s_2$, which is the substitution obtained by first applying s_2 to the terms of s_1 and then adding the ordered pairs from s_2 not occurring in s_1 . Example 2.10 below illustrates this said concept.

Example 2.10: Let $s_1 = \{f(X, Y)/Z\}$ and $s_2 = \{a/X, b/Y, c/W, d/Z\}.$

Then $s_1 \Delta s_2 = \{f(a, b)/Z, a/X, b/Y, c/W\}.$

Property 2: Composition of substitutions is associative, i.e.,

$$(s_1 \Delta s_2) \Delta s_3 = s_1 \Delta (s_2 \Delta s_3).$$

Example 2.11: To illustrate the associative property of composition in substitutions, let

$$\begin{aligned} s_1 &= \{f(Y)/X\} \\ s_2 &= \{a/Y\} \\ s_3 &= \{c/Z\} \\ \text{and } w &= P(X, Y, Z). \end{aligned}$$

$$\begin{aligned} \text{Here, } (s_1 \Delta s_2) &= \{f(a)/X, a/Y\} \\ (s_1 \Delta s_2) \Delta s_3 &= \{f(a)/X, a/Y, c/Z\} \end{aligned}$$

$$\begin{aligned} \text{Again, } (s_2 \Delta s_3) &= \{a/Y, c/Z\} \\ s_1 \Delta (s_2 \Delta s_3) &= \{f(a)/X, a/Y, c/Z\} \end{aligned}$$

$$\text{Therefore, } (s_1 \Delta s_2) \Delta s_3 = s_1 \Delta (s_2 \Delta s_3).$$

Property 3: Commutability fails in case of the substitutions, i.e.,

$$s_1 \Delta s_2 \neq s_2 \Delta s_1.$$

Example 2.12: We can illustrate the third property following the substitution sets used in example 2.11. Here,

$$\begin{aligned} (s_1 \Delta s_2) &= \{f(a)/X, a/Y\} \quad \text{and } (s_2 \Delta s_1) = \{a/Y, f(Y)/X\} \\ \text{So, } s_1 \Delta s_2 &\neq s_2 \Delta s_1. \end{aligned}$$

Definition 2.7: Given two predicates $P(t_1, t_2, \dots, t_n)$ and $P(r_1, r_2, \dots, r_n)$ and $s = \{t_i/r_i\}$ is a substitution set that on substitution in the predicates makes them identical (unifies them). Then the substitution set s is called a **unifier**. The **most general unifier (mgu)** is the simplest unifier g , so that any other unifier g' satisfies $g' = g \Delta s'$ for some substitution s' .

Example 2.13: For the predicates $P(X, f(Y), b)$ and $P(X, f(b), b)$, $g' = \{a/X, b/Y\}$ definitely is a unifier as it unifies the predicates to $P(a, f(b), b)$, but the mgu in this case is $g = \{b/Y\}$. It is to be noted further that a substitution $s' = \{a/X\}$ satisfies $g' = g \Delta s'$.

Definition 2.8: Let cl_i and cl_j be two clauses of the following form:

$$Cl_i \equiv A_{i1} \vee A_{i2} \vee \dots \vee A_{ix} \vee \dots \vee A_{im} \leftarrow B_{i1} \wedge B_{i2} \wedge \dots \wedge B_{il} \quad (2.17)$$

$$Cl_j \equiv A_{j1} \vee A_{j2} \vee \dots \vee A_{jm}' \leftarrow B_{j1} \wedge B_{j2} \wedge \dots \wedge B_{jy} \wedge \dots \wedge B_{jl}' \quad (2.18)$$

and P be the common literal present in the head of cl_i and body of cl_j . For instance, let for a substitution s

$$[A_{ix}]_s = [B_{jy}]_s = [P]_s$$

The **resolvent** of cl_i and cl_j , denoted by $R(cl_i, cl_j) = cl_{ij}$ (say), is computed as follows under the substitution $s = s_{ij}$, say:

$$Cl_{ij} = [(A_{i1} \vee A_{i2} \vee \dots \vee A_{i(x-1)} \vee A_{i(x+1)} \vee \dots \vee A_{im}) \vee (A_{j1} \vee A_{j2} \vee \dots \vee A_{jm}') \leftarrow (B_{i1} \wedge B_{i2} \wedge \dots \wedge B_{ii}) \wedge (B_{j1} \wedge B_{j2} \wedge \dots \wedge B_{j(y-1)} \wedge B_{j(y+1)} \wedge \dots \wedge B_{jl}')] \quad (2.19)$$

If cl_{ij} can be computed from the given cl_i and cl_j , we say that the clauses cl_i and cl_j are **resolvable**

Example 2.14: Given the following clauses

$$Cl_i: \text{Fly}(X) \leftarrow \text{Bird}(X). \quad (2.20)$$

$$Cl_j: \text{Bird}(\text{parrot}) \leftarrow. \quad (2.21)$$

$$Cl_{ij}: \text{Fly}(\text{parrot}) \leftarrow. \quad \text{where } s_{ij} = \{\text{parrot}/X\} \quad (2.22)$$

The clauses cl_i and cl_j in this example are resolvable with the substitution s_{ij} yielding the resolvent cl_{ij} .

Definition 2.9: If in a set of clauses there is at least one clause cl_j for each clause cl_i such that resolution holds on cl_i and cl_j , producing a resolvent cl_{ij} , then the set is called the **set of resolvable clauses**.

Example 2.15: In the following set of clauses each clause is resolvable with at least one other clause:

$$\text{Mother}(Y, Z) \leftarrow \text{Father}(X, Z), \text{Wife}(Y, X). \quad (2.23)$$

$$\text{Father}(r, l) \leftarrow. \quad (2.24)$$

$$\text{Wife}(s, r) \leftarrow. \quad (2.25)$$

Here, clause 2.23 is resolvable with clause number 2.24 and clause 2.25. It is an example of the set of resolvable clauses.

Now, we briefly outline select linear definite (SLD) resolutions.

2.2.3 SLD Resolution

To understand SLD resolution we first have to learn a few definitions.

Definition 2.10: A *definite program clause* is a clause of the form

$$A \leftarrow B_1, B_2, \dots, B_n,$$

which contains precisely one atom (viz. A) in its consequent (head) and a null, one or more literals in its body (viz. B_1 or B_2 or \dots or B_n).

Definition 2.11: A *definite program* is a finite set of definite program clauses.

Definition 2.12: A *definite goal* is a clause of the form

$$\leftarrow B_1, B_2, \dots, B_n.$$

i.e., a clause with an empty consequent.

Definition 2.13: *SLD resolution* stands for *SL resolution for definite clauses*, where *SL* stands for *resolution with linear selection function*.

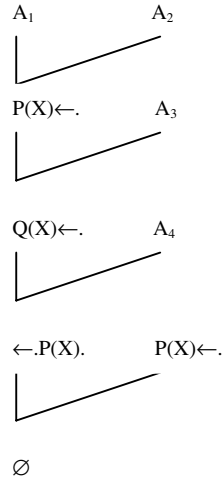


Fig. 2.1: The linear selection of clauses in the resolution tree.

Example 2.16: This example illustrates the linear resolution. Here, the following OR clauses (clauses connected by OR operator), represented by a set-like notation are considered.

Let $S = \{A_1, A_2, A_3, A_4\}$,

where $A_1 = P(X), Q(X) \leftarrow .$

$A_2 = P(X) \leftarrow Q(X).$

$$\begin{aligned}
A_3 &= Q(X) \leftarrow P(X). \\
A_4 &= \leftarrow P(X), Q(X). \\
\text{and goal} &= \leftarrow \neg P(X).
\end{aligned}$$

By linear selection, a resolution tree can be constructed as shown in Figure 2.1. It is clear from the tree that two clauses from the set $S_1 = S \cup \{\neg \text{Goal}\}$ are first used for resolution with a third clause from the same set S_1 . The process is continued until a null clause is generated. In the linear selection process, one clause, however, can be used more than once for resolution.

Definition 2.14: Let $S = \{cl_1, cl_2, \dots, cl_n\}$ be a set of resolvable clauses, and there exists one or more definite orders to pair-wise select the clauses for SLD resolution, without which the SLD resolution of all the n clauses fail to generate a resolvent. Under this circumstance, we say that an **orderly resolution** exists among the clauses in S .

The resolvent of clauses cl_i and cl_j is hereafter denoted by cl_{ij} or $R(cl_i, cl_j)$, where R represents a binary resolution operator. The **order of resolution** in $R(R(cl_i, cl_j), cl_k)$ is denoted by $i-j-k$ (or i, j, k) for brevity. It may be noted that $i-j \equiv j-i$ and $i-j-k \equiv k-i-j \equiv k-j-i$.

Example 2.17: Given the following clauses cl_1 through cl_3 , we would like to illustrate the principle of orderly resolution with these clauses.

$$\begin{aligned}
Cl_1: & \text{Paternal_uncle}(X, Y), \text{Maternal_uncle}(X, Y) \leftarrow \text{Uncle}(X, Y). \\
Cl_2: & \neg \text{Paternal_uncle}(n, a) \leftarrow. \equiv \leftarrow \text{Paternal_uncle}(n, a). \\
Cl_3: & \neg \text{Maternal_uncle}(n, a) \leftarrow. \equiv \leftarrow \text{Maternal_uncle}(n, a).
\end{aligned}$$

We now demonstrate two different orders of resolution, and show that the result is unique in both the cases. One of the orders of resolution could be 1-2-3. This is computed as follows:

$$\begin{aligned}
R(cl_1, cl_2): & \text{Maternal_uncle}(n, a) \leftarrow \text{Uncle}(n, a). \text{ with } s_{12} = \{n/X, a/Y\} \\
R(R(cl_1, cl_2), cl_3): & \leftarrow \text{Uncle}(n, a). \equiv \neg \text{Uncle}(n, a) \leftarrow.
\end{aligned}$$

An alternative order of resolution is 3-1-2. To compute this, we proceed as follows:

$$\begin{aligned}
R(cl_3, cl_1): & \text{Paternal_uncle}(n, a) \leftarrow \text{Uncle}(n, a). \text{ with } s_{31} = \{n/X, a/Y\} \\
R(R(cl_3, cl_1), cl_2): & \leftarrow \text{Uncle}(n, a). \equiv \neg \text{Uncle}(n, a) \leftarrow.
\end{aligned}$$

So, both the orderly resolutions return the same resolvent: $[\neg \text{Uncle}(n, a) \leftarrow.]$, which means n is not the uncle of a .

It is important to note that resolution of multiple clauses by different orders does not always return unique resolvents.

Definition 2.15: *If there exists only one definite order of resolution among a set of resolvable clauses, it is said to have **single sequence** of resolution. On occasions, we can obtain the same resolvent by taking a reversed order of resolution. For instance, $R(\dots(R(R(R(cl_1, cl_2), cl_3), cl_4) \dots cl_{n-1}), cl_n) = R \dots (R(R(cl_n, cl_{n-1}), \dots cl_3), cl_2), cl_1)$ is valid if the resolvents can be computed for each resolution. We, however, consider it a single order of resolution.*

Example 2.18: To understand the single sequence of resolution, let us take the following clauses:

$Cl_1: W(Z, X) \leftarrow P(X, Y), Q(Y, Z).$
 $Cl_2: P(a, b) \leftarrow S(b, a).$
 $Cl_3: S(Y, X) \leftarrow T(X).$

The resolution of clauses in the present context follows a definite order: 1-2-3 (or 3-2-1). It may be noted that

$$R(R(cl_1, cl_2), cl_3) = R(R(cl_3, cl_2), cl_1) = W(Z, a) \leftarrow Q(b, Z), T(a).$$

Consequently, a single sequence is maintained in the process of resolution of multiple program clauses.

Definition 2.16: *When in a set of resolvable clauses, resolution takes place following different orders, **multiple sequence** is said to be present.*

Example 2.19: The following clauses are taken to illustrate the multiple sequences in a set of resolvable clauses:

Given $Cl_1: S(Z, X) \leftarrow P(X, Y), Q(Y, Z), R(Z, Y).$
 $Cl_2: P(a, b) \leftarrow T(c, a), U(b).$
 $Cl_3: Q(b, c) \leftarrow V(b), M(c).$
 $Cl_4: R(Z, Y) \leftarrow N(Y), O(Z).$

Sequence 1: Order: 1-2-3-4.

$$\begin{aligned} R(cl_1, cl_2): & S(Z, a) \leftarrow Q(b, Z), R(Z, b), T(c, a), U(b). \mid s_{12}=\{a/X, b/Y\} \\ R(R(cl_1, cl_2), cl_3): & S(c, a) \leftarrow R(c, b), T(c, a), U(b), V(b), M(c). \mid s_{12,3}=\{c/Z\} \\ R(R(R(cl_1, cl_2), cl_3), cl_4): & S(c, a) \leftarrow T(c, a), U(b), V(b), M(c), N(b), \\ & O(c). \mid s_{12,3,4}=\{c/Z, b/Y\} \end{aligned}$$

Sequence 2: Order: 3-1-2-4.

$$\begin{aligned} R(cl_3, cl_1): S(c, X) \leftarrow P(X, b), R(c, b), V(b), M(c). \mid s_{31} = \{c/Z, b/Y\} \\ R(R(cl_3, cl_1), cl_2): \\ S(c, a) \leftarrow T(c, a), U(b), P(a, b), R(c, b), V(b), M(c). \mid s_{31,2} = \{a/X\} \\ R(R(R(cl_3, cl_1), cl_2), cl_4): \\ S(c, a) \leftarrow T(c, a), U(b), V(b), M(c), N(b), O(c). \mid s_{31,2,4} = \{c/Z, b/Y\} \end{aligned}$$

Sequence 3: Order: 1-2-4-3.

$$\begin{aligned} R(R(cl_1, cl_2), cl_4): \\ S(Z, a) \leftarrow Q(b, Z), T(c, a), U(b), N(b), O(Z). \mid s_{12,4} = \{b/Y\} \\ R(R(R(cl_1, cl_2), cl_4), cl_3): \\ S(c, a) \leftarrow T(c, a), U(b), V(b), M(c), N(b), O(c). \mid s_{1,2,4,3} = \{c/Z\} \end{aligned}$$

The results of the above computation reveal that

$$\begin{aligned} & R(R(R(cl_1, cl_2), cl_3), cl_4) \\ &= R(R(R(cl_3, cl_1), cl_2), cl_4) \\ &= R(R(R(cl_1, cl_2), cl_4), cl_3) \\ &= S(c, a) \leftarrow T(c, a), U(b), V(b), M(c), N(b), O(c). \end{aligned}$$

Consequently, multiple order exists in the resolution of clauses.

Readers may please note that resolution between cl_2 and cl_3 , cl_3 and cl_4 , and cl_2 and cl_4 are not possible.

Definition 2.17: Let $S = \{cl_1, cl_2, \dots, cl_n\}$ be a set of resolvable clauses and the cl_i s are ordered in a manner that cl_i and cl_{i+1} are resolvable for $i = 1, 2, \dots, (n-1)$. If cl_n is also resolvable with cl_1 we call it **circular resolution**. Circular resolution is not allowed as it invites multiple resolution between two clauses.

Example 2.20: Consider the following propositional clauses:

$$\begin{aligned} Cl_1: Q \leftarrow P. \\ Cl_2: R \leftarrow Q. \\ Cl_3: P \leftarrow R. \end{aligned}$$

$$\begin{aligned} \text{Let } Cl_{12} &= R(Cl_1, Cl_2) \\ &= R \leftarrow P. \end{aligned}$$

However, evaluation of $R(Cl_{12}, Cl_3)$ cannot be performed as two resolutions are applicable between the clauses, which is not allowed.

Definition 2.18: Let S be a set of resolvable clauses such that their pair-wise selection for SLD resolution from S is random. Under this condition, S is called the set of **order-less** or **order-independent** clauses.

Example 2.21: Let S be the set of the following clauses:

$Cl_1: U \leftarrow P, Q, R.$
 $Cl_2: S, M, P \leftarrow V.$
 $Cl_3: Q \leftarrow W, M, T.$
 $Cl_4: T \leftarrow U, S.$

In this example, we attempt to resolve each clause with others.

$Cl_{12}: U, S, M \leftarrow V, Q, R.$
 $Cl_{13}: U \leftarrow P, R, W, M, T.$
 $Cl_{14}: T \leftarrow P, Q, R, S.$
 $Cl_{23}: S, P, Q \leftarrow V, W, T.$
 $Cl_{34}: Q \leftarrow W, M, U, S.$
 $Cl_{24}: M, P, T \leftarrow V, U.$

As each of the clauses is resolvable with each other, order-less condition for resolution holds here.

It is important to note that order-less resolution is not valid even for propositional clauses. The example below provides an insight to this problem.

2.3 Concurrency in Resolution

To speed up the execution of logic programs, we take a look at the possible parallelism/ concurrency in the resolutions involved in the program.

2.3.1 Preliminary Definitions

Definition 2.19: If S includes multiple ordered sequence of clauses for SLD resolution and for each such sequence the final resolvent is identical then the clauses in S are **concurrently resolvable**. Under this case, resolution of all the clauses can be done concurrently yielding the same resolvent.

Example 2.22: In this example we consider the concurrent resolution of propositional clauses.

$Cl_1: R \leftarrow P, Q.$
 $Cl_2: P \leftarrow S.$
 $Cl_3: Q \leftarrow T.$

$$Cl_4 : T \leftarrow U.$$

Sequence 1: Order: 1-2-3-4.

$$\begin{aligned} R(Cl_1, Cl_2) &: R \leftarrow Q, S. \\ R(R(Cl_1, Cl_2), Cl_3) &: R \leftarrow T, S. \\ R(R(R(Cl_1, Cl_2), Cl_3), Cl_4) &: R \leftarrow U, S. \end{aligned}$$

Sequence 2: Order: 3-1-4-2.

$$\begin{aligned} R(Cl_3, Cl_1) &: R \leftarrow P, T. \\ R(R(Cl_3, Cl_1), Cl_4) &: R \leftarrow P, U. \\ R(R(R(Cl_3, Cl_1), Cl_4), Cl_2) &: R \leftarrow U, S. \end{aligned}$$

Sequence 3: Order: 3-1-2-4.

$$\begin{aligned} R(R(Cl_3, Cl_1), Cl_2) &: R \leftarrow S, T. \\ R(R(R(Cl_3, Cl_1), Cl_2), Cl_4) &: R \leftarrow U, S. \end{aligned}$$

Sequence 4: Order: 3-4-1-2.

$$\begin{aligned} R(Cl_3, Cl_4) &: Q \leftarrow U. \\ R(R(Cl_3, Cl_4), Cl_1) &: R \leftarrow P, U. \\ R(R(R(Cl_3, Cl_4), Cl_1), Cl_2) &: R \leftarrow U, S. \end{aligned}$$



Fig. 2.2: Concurrent resolution.

Since all the four sequences yield the same resolvent, the given clauses can be resolved concurrently. Figure 2.2 thus concurrently resolves four clauses and yields a resulting clause: $R \leftarrow U, S$. But how can we concurrently resolve these four clauses? A simple answer to this is to drop the common literals from the head of one clause and the body of another clause, and then group the remaining literals in the heads and in the body of the participating clauses. We employed this principle in Figure 2.2.

Definition 2.20: *In case multiple sequences exist in orderly resolution and the resolvent is not unique, then the substitutions used in resolution in each pair of*

clauses may be propagated downstream in the process of SLD resolution. The composition of the substitution sets for every two sequential substitutions is also carried forward along the SLD tree until the final resolvent is obtained. The final substitution may now be used as the instantiation space of the resolvent and the resulting clause thus generated for each such sequence is compared. In case the instantiated resolvent generated following multiple sequences yields a unique result then the clauses in S are also called **concurrently resolvable**. The final substitution set is called the **deferred substitution set**.

Example 2.23: To illustrate the above situation

let

$Cl_1: R(Z, X) \leftarrow P(X, Y), Q(Y, Z).$

$Cl_2: P(a, b) \leftarrow S(b, a).$

$Cl_3: Q(b, c) \leftarrow T(c, b).$

$Cl_4: T(Z, Y) \leftarrow U(X, Y).$

Sequence 1: Order: 1-2-3-4.

$$\begin{aligned} R(Cl_1, Cl_2): R(Z, a) &\leftarrow Q(b, Z), S(b, a). \mid s_{12}=\{a/X, b/Y\} \\ R(R(Cl_1, Cl_2), Cl_3): R(c, a) &\leftarrow T(c, b), S(b, a). \mid s_{12,3}=\{c/Z\} \\ R(R(R(Cl_1, Cl_2), Cl_3), Cl_4): R(c, a) &\leftarrow S(b, a), U(X, b). \mid s_{12,3,4}=\{c/Z, b/Y\} \end{aligned}$$

\therefore Composition of the substitutions, $s_{12} \Delta s_{12,3} = \{a/X, b/Y, c/Z\}$ and the final composition of the substitutions, $s_{12} \Delta s_{12,3} \Delta s_{12,3,4} = \{a/X, b/Y, c/Z\}$.

Sequence 2: Order: 3-1-2-4.

$$\begin{aligned} R(Cl_3, Cl_1): R(c, X) &\leftarrow P(X, b), T(c, b). \mid s_{31}=\{b/Y, c/Z\} \\ R(R(Cl_3, Cl_1), Cl_2): R(c, a) &\leftarrow T(c, b), S(b, a). \mid s_{31,2}=\{a/X\} \\ R(R(R(Cl_3, Cl_1), Cl_2), Cl_4): R(c, a) &\leftarrow S(b, a), U(X, b). \mid s_{31,2,4}=\{c/Z, b/Y\} \end{aligned}$$

\therefore Composition of the substitutions, $s_{31} \Delta s_{31,2} = \{a/X, b/Y, c/Z\}$ and the final composition of the substitutions, $s_{31} \Delta s_{31,2} \Delta s_{31,2,4} = \{a/X, b/Y, c/Z\}$.

Sequence 3: Order: 3-4-1-2

$$\begin{aligned} R(Cl_3, Cl_4): Q(b, c) &\leftarrow U(X, b). \mid s_{34}=\{b/Y, c/Z\} \\ R(R(Cl_3, Cl_4), Cl_1): R(c, X) &\leftarrow P(X, b), U(X, b). \mid s_{34,1}=\{b/Y, c/Z\} \\ R(R(R(Cl_3, Cl_4), Cl_1), Cl_2): R(c, a) &\leftarrow S(b, a), U(a, b). \mid s_{34,1,2}=\{a/X\} \end{aligned}$$

\therefore Composition of the substitutions, $s_{34} \Delta s_{34,1} = \{b/Y, c/Z\}$ and the final composition of the substitutions, $s_{34} \Delta s_{34,1} \Delta s_{34,1,2} = \{a/X, b/Y, c/Z\}$.

Now, if we compute the deferred substitution set for the three sequences, we find it to be equal, the value of which is given by $s = \{a/X, b/Y, c/Z\}$. When the resolvents are instantiated by this deferred substitution set, they become equal and the final resolvent is given by $R(c, a) \leftarrow S(b, a), U(a, b)$.

2.3.2 Types of Concurrent Resolution

There are three types of concurrent resolution:

- a) Concurrent resolution of a rule with facts,
- b) Concurrent resolution of multiple rules,
- c) Concurrent resolution of both multiple rules and facts.

Various well-known types of parallelisms are involved in the concurrent resolutions.

Definition 2.21: When the predicates of a rule are attempted for matching with predicates contained in the facts, **concurrent resolution of the rule with facts** is said to take place.

It can occur in two ways. In the first case, the literals present in the body part of a clause (AND literals) may be searched against the literals present in the heads of the available facts, which is a special case of **AND-Parallelism**.

Example 2.24: To illustrate AND-parallelism let us consider the following clauses:

$$\text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y), \text{Married_to}(X, Z). \quad (2.26)$$

$$\text{Father}(r, n) \leftarrow. \quad (2.27)$$

$$\text{Married_to}(r, t) \leftarrow. \quad (2.28)$$

Here, predicates in the heads of the facts given by the clauses (2.27) and (2.28) are concurrently resolved with the predicates in the body of the rule given by (2.26) yielding the resolvent

$$\text{Mother}(t, n) \leftarrow.$$

Again, when a literal present in the body of one rule may be searched concurrently against the literals present in the heads of more than one fact, **OR-Parallelism** is invoked.

Example 2.25: We can illustrate OR-parallelism with the help of the following clauses:

$$\text{Son}(X, Y) \leftarrow \text{Father}(Y, X). \quad (2.29)$$

$$\text{Father}(r, n) \leftarrow . \quad (2.30)$$

$$\text{Father}(n, a) \leftarrow . \quad (2.31)$$

Here, the variables present in the body of the rule given by (2.29) can be matched concurrently with the arguments in the heads of the facts given by 2.30 and 2.31.

Definition 2.22: *When more than one rule is resolved concurrently in a given set of resolvable clauses, we say that **concurrent resolution of multiple rules** has taken place.*

Example 2.26: The following clauses are considered to explain the concurrent resolution of multiple rules:

$$\text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y), \text{Wife}(Z, X). \quad (2.32)$$

$$\text{Wife}(Y, X) \leftarrow \text{Female}(Y), \text{Married_to}(Y, X). \quad (2.33)$$

$$\text{Married_to}(X, Y) \leftarrow \text{Marries}(X, Y). \quad (2.34)$$

Here, the concurrent resolution can take place between the rules 2.32 and 2.33 in parallel with the rules 2.33 and 2.34. Moreover, the above three rules can be concurrently resolved together yielding the resolvent

$$\text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y), \text{Female}(Z), \text{Marries}(X, Z). \quad (2.35)$$

A special kind of parallelism, known as **Stream-parallelism** can be encountered while discussing concurrent resolution of multiple rules. It is explained with the following example.

Example 2.27: Let us consider the following clauses:

$$\text{Integer}(X+1) \leftarrow \text{Integer}(X). \quad (2.36)$$

$$\text{Evaluate_square}(Z.Z) \leftarrow \text{Integer}(Z). \quad (2.37)$$

$$\text{Print}(Y) \leftarrow \text{Evaluate_square}(Y). \quad (2.38)$$

$$\text{Integer}(0) \leftarrow . \quad (2.39)$$

Here, the resolvent obtained by resolving the rule (2.36) and the fact (2.39) is propagated to resolve the rule (2.37). The resolvent thus obtained is resolved further with (2.38). The process is repeated in a streamline for the integer sequence 0, 1, 2, . . . up to infinity. After one result is printed, the pipeline becomes busy rest of the time, and resolution of three pairs of clauses takes place concurrently.

Definition 2.23: *If more than one rule is resolved concurrently with more than one fact, **concurrent resolution of both multiple rules and facts** takes place.*

Example 2.28: To illustrate the concurrent resolution of multiple rules and facts, let us take the following clauses:

$$R(Z, X) \leftarrow P(X, Y), Q(Y, Z). \quad (2.40)$$

$$P(a, b) \leftarrow. \quad (2.41)$$

$$Q(b, c) \leftarrow. \quad (2.42)$$

$$S(U, V), T(U, V) \leftarrow R(U, V). \quad (2.43)$$

$$\leftarrow S(d, e). \quad (2.44)$$

$$\leftarrow T(d, e). \quad (2.45)$$

Here, concurrent resolution can take place in several ways. At first, the rules given by (2.40) and (2.43) can resolve generating the resolvent $S(Z, X), T(Z, X) \leftarrow P(X, Y), Q(Y, Z)$. Again, the rule (2.40) is resolved with the facts given by (2.41) and (2.42) in parallel while the rule (2.43) is resolved with the given facts (2.44) and (2.45). The resolvents in these two cases are, respectively:

$$R(c, a) \leftarrow Q(b, c). \text{ and } \leftarrow R(d, e).$$

When the predicates present in the body of one rule also occur in the head of a second rule, the latter and the former rules are said to be in pipeline. Rules in pipeline are resolvable. But in case there exists a matching ground clause for the common literals of both the rules, it is preferred to resolve the ground clause with either of (or both) the rules. The Petri-net model for extended logic programming that we would like to introduce shortly is designed based on the above concept.

The following observations can be made from the last example.

- 1) Resolution of a rule with one or more facts provides a scope for yielding intermediate ground inferences. For instance, the rule (2.40) in Example 2.29 when resolved with (2.41) and (2.42) yields a ground intermediate $R(c, a) \leftarrow Q(b, c)$.
- 2) Resolution of two or more rules yield new rules containing literals with renamed variables. The effort in doing so, on many occasions, may be fruitless. For instance, if rule (2.40) and (2.43) were resolved, a new rule would be generated with no further benefits of re-resolving the resulting rule with available facts.
- 3) In case there exist concurrently resolvable groups of clauses, where each group contains a rule and a few facts, then the overall computational speed of the system can be significantly improved.

Example 2.29 below provides an insight to this issue.

Example 2.29: Let us take the following clauses:

$$R(Z, X) \leftarrow P(X, Y), Q(Y, Z). \quad (2.46)$$

$$P(a, b) \leftarrow. \quad (2.47)$$

$$Q(b, c) \leftarrow. \quad (2.48)$$

$$S(U, V), T(U, V) \leftarrow R(U, V). \quad (2.49)$$

$$\leftarrow S(c, a). \quad (2.50)$$

$$\leftarrow T(c, a). \quad (2.51)$$

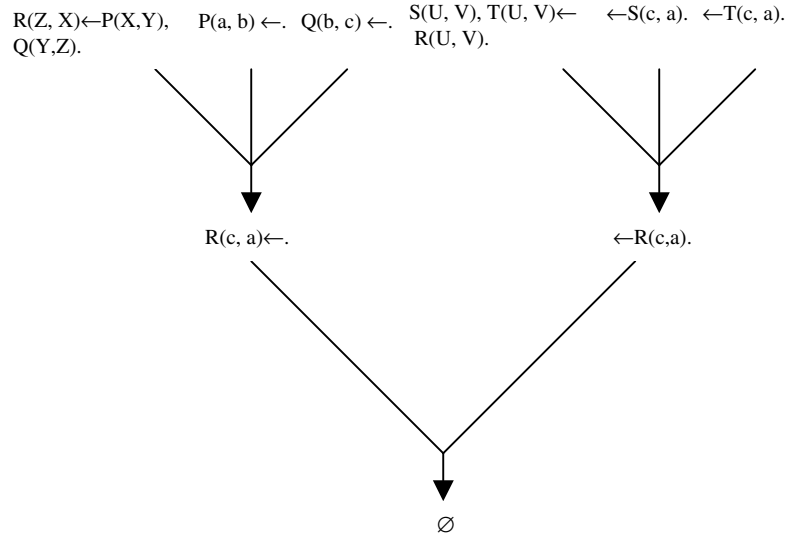


Fig 2.3: A graph illustrating concurrency in resolution.

Unlike example (2.28), where the resulting resolvents after concurrent resolution of two groups of clauses could not participate in further resolution, here the resolvents due to concurrent resolution of (2.46-2.48) and (2.49-2.51) can also take part in the resolution game, resulting in a null clause. To have an idea of speed-up, we construct a graph (Fig. 2.3) indicating the concurrency in resolution.

It is apparent from the above graph that the concurrent resolution of clauses (2.46-2.48) and (2.49-2.51) can take place in one unit time, and the resolution of the resulting clauses require one unit time. Thus, the time taken for execution of the logic program on a parallel engine is two unit times. The same problem, if solved by SLD tree, takes as many as five unit times to perform five resolutions of binary clauses (Fig. 2.4).

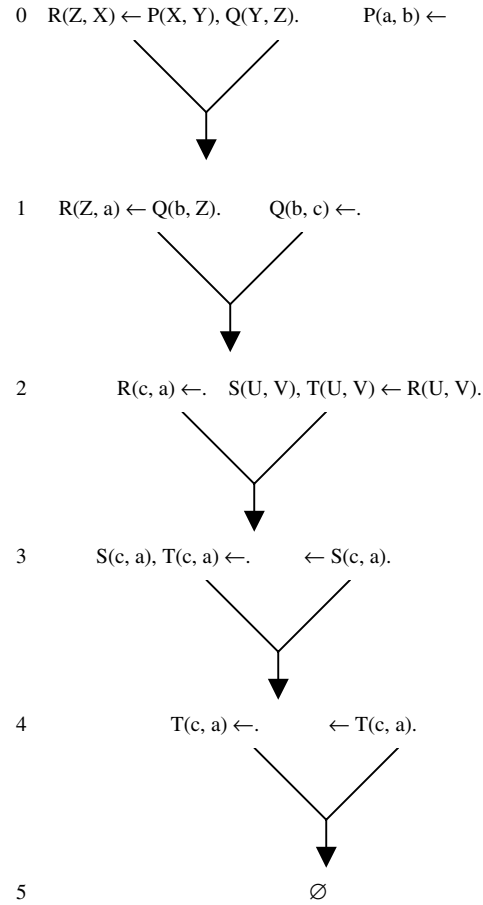


Fig. 2.4: The SLD Tree.

The example shows that definitely there is a scope in speed-up due to concurrent resolution at the cost of additional expenses for hardware resources.

The most important problem in concurrent resolution is the identification of the clauses that participate in the resolution process. When there exist groups of concurrently resolvable clauses, search cost to detect the participating clauses in each group sometimes is too high to be amenable in real time. A specialized data structure, capable of performing concurrent resolution of multiple groups of clauses, is recommended to handle the problem. In fact, we are in search of a suitable structure where participating facts and rules in one group of resolvable clauses can be represented by neighborhood structural units. The search cost

needed in concurrent resolution thus can be saved by the above-mentioned data structure.

Petri nets, which have proved themselves successful in solving many complex problems of knowledge engineering, can equally be used in the present context to efficiently handle the problems of concurrent resolution. For example, let us consider a clause ' $P(X, Y), Q(X, W) \leftarrow R(X, Y), S(Y, W).$ ' which is represented in a Petri net by a transition and four associated places, where P and Q are represented by output places, and R and S are denoted by input places of the transition. The argument of each literal in a rule is represented by a specialized function, called arc function, which is associated with the arc connecting the transition with the respective places. The arc functions are needed for generation of variable bindings in the process of resolution of clauses. If ' $\neg P(a, b) \leftarrow.$ ', ' $R(a, b) \leftarrow.$ ' and ' $S(b, c) \leftarrow.$ ' are supplied as additional facts then they could be mapped in the places connected with the proposed transition, and the arguments $\neg\langle a, b \rangle$, $\langle a, b \rangle$, and $\langle b, c \rangle$ of the facts are saved as tokens of the respective places P, R and S. Such neighborhood mapping of the rule and facts described in Figure 2.5 help concurrent resolution with no additional time for searching the concurrently resolvable clauses.

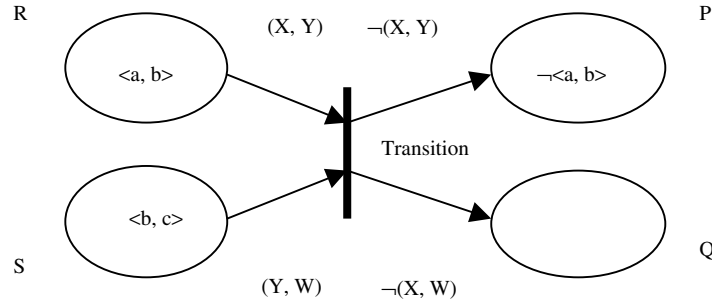


Fig. 2.5: Mapping of a rule on to a Petri net.

Reasoning in a logic program using Petri nets was first proposed by Murata [10]. In this chapter, we extended Murata's model by the following counts:

- ◆ In Murata's model arc functions associated with the arcs of a Petri net are positive irrespective of the type² of the arcs. The model to be proposed shortly, however, assigns a positive sign to the arc function attached with a place-to-transition connective arc, and a negative sign

² The directed arcs in a Petri net denote connectivity from i) places to transitions, and ii) transitions to places, and thus are of two basic types.

to the arc function attached with a transition-to-place connective arc. The attachment of sign with the arc functions facilitates the scope of matching of signed tokens of the respective places with the arc functions of the connected arcs following the formalisms of predicate logic.

- ◆ Unlike Murata's model, where the arguments of body-less clauses also were represented as arc functions, in the present model these are represented as tokens of the appropriate places. Thus, in the present model, we can save additional transition firings for all those arc functions corresponding to the body-less clauses.
- ◆ The extension of Murata's model presented here allows AND-, OR-, Unification-, and Stream-parallelism in a logic program.

2.4. Petri Net Model for Concurrent Resolution

2.4.1 Extended Petri Net

Definition 2.24: An *extended Petri net (EPN)*, which will be used here for reasoning with a First Order Logic (FOL) program, is a 9-tuple, given by

$$EPN = \{P, Tr, D, f, m, A, a, I, O\}$$

where

$P = \{p_1, p_2, \dots, p_m\}$ is a set of places;

$Tr = \{tr_1, tr_2, \dots, tr_n\}$ is a set of transitions;

$D = \{d_1, d_2, \dots, d_m\}$ is a set predicates;

$P \cap Tr \cap D = \emptyset$; Cardinality of P = Cardinality of D ;

$f: D \rightarrow P^\infty$ represents a mapping from the set of predicates to the set of places;

$m: P \rightarrow \langle x_1, \dots, y_i, X, \dots, Y, f, \dots, g \rangle$ is an association function, represented by the mapping from places to terms, which may include signed constant(s) like x_i, \dots, y_i , variable(s) like X, \dots, Y and function f, \dots, g of variables; it is usually referred to as tokens of a given place; $A \subseteq (P \times Tr) \cup (Tr \times P)$ is the set of arcs, representing the mapping from the places to the transitions and vice versa;

$a: A \rightarrow (X, Y, \dots, Z)$ is an association function of the arcs, represented by the mapping from the arcs to terms. For arcs $A \in (P \times Tr)$ the arc functions a are positively signed, while for arcs $A \in (Tr \times P)$ the arc functions a are negatively signed;

$I: Tr \rightarrow P^\infty$ is a set of input places, represented by the mapping from the transitions to their input places;

$O: Tr \rightarrow P^\infty$ is a set of output places, represented by the mapping from the transitions to their output places.

Example 2.30: Mapping of the above parameters onto an EPN is illustrated (Fig. 2.6) in this example with the following FOL clauses:

$$\text{Son}(Y, Z), \text{Daughter}(Y, Z) \leftarrow \text{Father}(X, Y), \text{Wife}(Z, X). \quad (2.52)$$

$$\text{Father}(r, l) \leftarrow. \quad (2.53)$$

$$\text{Wife}(s, r) \leftarrow. \quad (2.54)$$

$$\neg \text{Daughter}(l, s) \leftarrow. \quad (2.55)$$

Here, $P = \{p_1, p_2, p_3, p_4\}$;

$\text{Tr} = \{\text{tr}_1\}$;

$D = \{d_1, d_2, d_3, d_4\}$ with $d_1 = \text{Father}$, $d_2 = \text{Wife}$, $d_3 = \text{Son}$ and $d_4 = \text{Daughter}$;

$f(\text{Father}) = p_1$, $f(\text{Wife}) = p_2$, $f(\text{Son}) = p_3$, $f(\text{Daughter}) = p_4$;

$m(p_1) = \langle r, l \rangle$, $m(p_2) = \langle s, r \rangle$, $m(p_3) = \langle \emptyset \rangle$, $m(p_4) = \neg \langle l, s \rangle$ initially and can be computed subsequently through unification of predicates in the process of resolution of clauses;

$A = \{A_1, A_2, A_3, A_4\}$, and

$a(A_1) = (X, Y)$, $a(A_2) = (Z, X)$, $a(A_3) = \neg(Y, Z)$, $a(A_4) = \neg(Y, Z)$ are the arc functions;

$I(\text{tr}_1) = \{p_1, p_2\}$, and $O(\text{tr}_1) = \{p_3, p_4\}$.

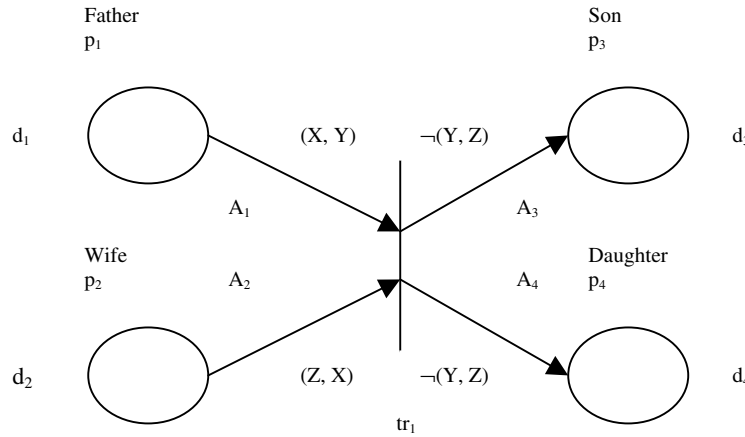


Fig. 2.6: Parameters of an EPN used to represent knowledge in FOL.

It is to be noted that if-then operator of the knowledge has been represented in the figure by tr_1 and the antecedent-consequent pairs of knowledge have been denoted by input (I)-output (O) places of tr_1 . Moreover, the arguments of the predicates have been represented by arc functions.

For computing tokens at a place, the transition associated with the place needs to be enabled and fired.

Definition 2.25: A transition tr_i is **enabled** if the conditions listed below are jointly satisfied:

- i) all places excluding at most one place associated with the transition tr_i possess properly signed tokens, i.e., positively signed tokens for the input places and negatively signed tokens for the output places, and
- ii) the variables in the arc functions have consistent bindings with respect to the tokens of the associated places. The consistent binding means that each variable in the arc functions should have a common value.

Definition 2.26: A transition, which is enabled first-time **fires**. An enabled transition that already fired can also **fire** if the value of consistent bindings of all the arc function variables with respect to the transition is different from the existing bindings obtained earlier. On firing of a transition, new tokens are generated at a place that did not participate in the generation of consistent variable bindings. The value of the token is determined by the ordered value of the variables in the arc function associated with the concerned place, and the sign of the token will be opposite to the sign of the said arc function.

Example 2.31: The EPN shown in Fig. 2.6 is enabled because the transition tr_1 here has four input/ output places out of which three places contain properly signed tokens, and the variable in the arc functions are consistent. To test the consistency we list below the value of the variables in each arc function by matching them with their associated places.

$$\begin{aligned} a(A_1) &= (X, Y) = \langle r, l \rangle \text{ on matching with tokens of place } p_1. \\ a(A_2) &= (Z, X) = \langle s, r \rangle \text{ on matching with tokens of place } p_2. \\ a(A_4) &= \neg(Y, Z) = \neg \langle l, s \rangle \text{ on matching with tokens of place } p_4. \end{aligned}$$

Thus, consistent binding is given by $X = r$, $Y = l$, and $Z = s$. The transition fires because it is enabled and it did not fire earlier. Since place p_3 did not participate in the construction of consistent bindings, a token is inserted in place p_3 on firing of the transition. The token is computed by taking the ordered variables in arc function A_3 and complementing the sign of the arc function. The token thus obtained at place p_3 is given by $\neg(\neg(Y, Z)) = \langle l, s \rangle$.

2.4.2 Algorithm for Concurrent Resolution

Notations used in the algorithm for automated reasoning are, in order, as follows:

Current-bindings (c-b) denote the set of instantiation of all the variables associated with the transitions.

Used-bindings (u-b) denote the set of union of the current-bindings up to the last transition firing.

Properly signed token means tokens with proper signs, i.e., positive tokens for input places and negative tokens for output places of a transition.

Inactive arc functions represent the arc functions associated with a transition, which do not participate in the process of generation of consistent bindings of variables.

Inert place denotes a place that did not participate in the generation of current bindings of a transition.

The algorithm for automated reasoning to be presented shortly allows concurrent firing of multiple transitions. The algorithm in each pass checks the enabling conditions of all the transitions. If one or more transitions are found enabled, the *mgus*, here referred to as current-bindings for each transition is searched against the union of the preceding *mgus*, called used-bindings of the said transition. If the current-bindings are not members of the respective used-bindings, then the enabled transitions are fired concurrently. The tokens for the inert places are then computed following the current-bindings of the fired transitions.

The process of transition firing continues until no further transition is ready for firing.

Procedure Automated-reasoning

Begin

For each transition **do**

Par Begin

used-bindings:= Null;

Flag:= true; // transition not fireable.//

Repeat

If at least all minus one number of the input plus
the output places of the transition possess *properly*
signed tokens

Then do

Begin

determine the set: current-bindings of all the
variables associated with the transition;

If (a non-null binding is available for all the
variables) AND
(current-bindings is not a member of used-bindings)

Then do Begin

Fire the transition and send tokens to the *inert place*
using the set current-bindings and following the

```

    inactive arc function with a presumed opposite sign;
    Update used-bindings by taking union with
    current-bindings;
    Flag:= false; //record of transition firing//
    Increment no-of-firing by 1;
  End
Else Flag:= true;
End;
Until no transition fires;
Par End;
End.

```

Trace of the Algorithm

The Petri net shown in Figure 2.7 is constructed with a set of rules (2.56-2.64). The reasoning algorithm presented earlier is then invoked and the trace of the algorithm thus obtained is presented in Table 2.1. It is clear from the table that the current-bindings (c-b) are not members of used bindings (u-b) in the first two firing criteria testing (FCT) cycles. Therefore, flag = 0. Thus, following the algorithm, transitions tr_1 and tr_2 both fire concurrently. In the third cycle current-bindings become members of used-bindings, and, consequently, flag = 1; so no firing takes place during the third cycle. Further, number of transition in the Petri net (vide Fig. 2.7) being 2 only, control exits the repeat-until loop in procedure Automated-reasoning after two FCT cycles.

Rules:

Father (Y, Z), Uncle (Y, Z) \leftarrow Father (X, Y), Grandfather (X, Z). (2.56)

Paternal_uncle (X, Y), Maternal_uncle (X, Y) \leftarrow Uncle (X, Y). (2.57)

Father (r, n) \leftarrow . (2.58)

Father (r, d) \leftarrow . (2.59)

\neg Father (d, a) \leftarrow . (2.60)

Grandfather (r, a) \leftarrow . (2.61)

\neg Paternal_uncle (n, a) \leftarrow . (2.62)

\neg Maternal_uncle (n, a) \leftarrow . (2.63)

\neg Maternal_uncle(d, a) \leftarrow . (2.64)

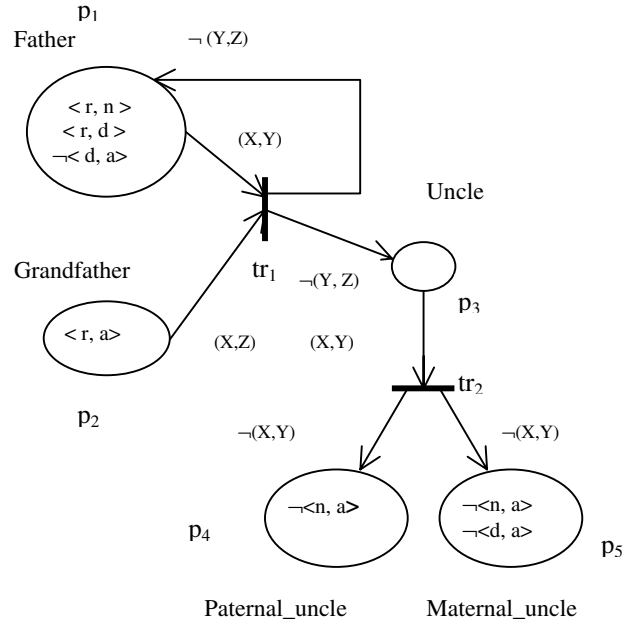


Fig. 2.7: An illustrative Petri net with initial assignment of tokens used to verify the procedure automated-reasoning.

Table 2.1: Trace of the algorithm on example net of Fig. 2.7

Time slot	Tran.	Set of c-b	Set of u-b	Flag=0, if c-b \notin u-b =1, if c-b \in u-b $\neq \{\phi\}$ or c-b= $\{\phi\}$
First cycle	tr ₁	{r/x,d/y,a/z}	{ $\{\phi\}$ }	0
	tr ₂	{n/x,a/y}	{ $\{\phi\}$ }	0
Second cycle	tr ₁	{r/x,n/y,a/z}	{ {r/x,d/y,a/z} }	0
	tr ₂	{d/x,a/y}	{ {n/x, a/y} }	0
Third cycle	tr ₁	{r/x, d/y,a/z}/ {r/x,n/y,a/z}	{ {r/x,d/y,a/z}, {r/x,n/y,a/z} }	1
	tr ₂	{n/x, a/y}/ {d/x, a/y}	{ {n/x, a/y}, {d/x, a/y} }	1

2.5 Performance Analysis of Petri Net-Based Models

In this section we outline two important issues: the *speed-up factor* and the *resource utilization rate* of the proposed algorithm when realized on a parallel architecture.

2.5.1 The Speed-up

A complexity analysis of a logic program of n clauses comprised of predicates of arity p reveals that the time T_u required for execution of the program by SLD-resolution on a uniprocessor architecture is given by

$$T_u = O(p.n). \quad (2.65)$$

The product $(p.n)$ in the order of complexity appears because of SLD-resolution of n clauses with p sequential matching of arguments of predicates involved in the resolution process.

The same program comprised of m_1, m_2, \dots, m_k number of concurrently resolvable clauses suppose is executed⁴ on a pipelined (multiprocessor) architecture, capable of resolving $\max \{m_i; 1 \leq i \leq k\}$ number of clauses in a unit time. Let m_i includes s_i number of supplied clauses and d_i number of derived clauses. Thus $\Sigma m_i = \Sigma s_i + \Sigma d_i$. Under this circumstance the total computational time T_m for execution of the logic program is given by

$$\begin{aligned} T_m &= O[p(n - (s_1 + s_2 + s_3 + \dots + s_k) - 1 + 1 \times k)]. \\ &= O[p(n - \sum_{i=1}^k s_i + k - 1)] \\ &\approx O[p(n - \sum_{i=1}^k s_i + k)] \end{aligned} \quad (2.66)$$

When Σs_i approaches Σm_i , T_m is maximum.

The above result presumes a k -stage pipeline of the k -sets of concurrently resolvable clauses. If the k -sets of clauses are independent, then the concurrent resolution of all the k -set of clauses can be accomplished within a unit time, and thus the computational complexity further reduces to $O[p(n - \sum_{i=1}^k s_i)]$. Thus, irrespective of a program, it can easily be ascertained that the computational time T_m of a typical logic program always lies in the interval:

⁴ The k sets of concurrent clauses here is assumed to be resolved in sequence, i.e., resolution of one concurrent set of clause is dependent on a second set, and thus all sets of concurrent clauses cannot be resolved in parallel.

$$O[p(n - \sum_{i=1}^k s_i)] \leq T_m \leq O[p(n - \sum_{i=1}^k s_i + k)] \quad (2.67)$$

Thus, *speed-up*⁵ in the worst case is found to be

$$\begin{aligned} S &= T_u / T_m \\ &= (p \cdot n) / [p(n - \sum_{i=1}^k s_i + k)] \\ &= n / [n - \sum_{i=1}^k s_i + k] \end{aligned} \quad (2.68)$$

In case all the n number of program clauses are exhausted by resolution, i.e., $\sum_{i=1}^k m_i$ approaches n , then $S = S_{\max}$ is maximized, and the speed-up factor S_{\max} reduces to

$$S_{\max} = n / k. \quad (2.69)$$

The last expression reveals that smaller is the k , larger is the S_{\max} . The best case corresponds to $k=1$, when there is a single set of concurrently resolvable clauses. But since $\sum_{i=1}^k s_i = n$ and $k=1$ in the present context, $\sum_{i=1}^k s_i = \sum_{i=1}^1 s_i = s_1 = n$, which means all the n set of clauses are resolvable together. Consequently, the speed-up factor is n .

On the other extreme end, when $k = n$, i.e., there are s_1, s_2, \dots, s_n number of concurrently resolvable sets of clauses, then $S_{\max} = n / n = 1$, and there is no speed-up. In fact, this case corresponds to typical SLD-resolution and the number of clauses $s_1 = s_2 = \dots = s_n = 2$.

2.5.2 The Resource Utilization Rate

Let us assume that the number of resources available for concurrent resolution in the present context is $\max \{s_k : 1 \leq k \leq n\}$. Thus, maximum degree of parallelism [4] P is given by

$$P = \max \{s_k : 1 \leq k \leq n\}. \quad (2.70)$$

⁵ In case of unification-parallelism, T_m reduces to $O[(n - \sum_{i=1}^k s_i + k)]$ and, consequently, the worst-case speed-up factor becomes $S_{\max} = (p \cdot n) / (n - \sum_{i=1}^k s_i + k)$.

The average degree of parallelism P_{av} is defined below following Hwang and Briggs [4] as

$$P_{av} = (\sum_{i=1}^k s_i) / k. \quad (2.71)$$

The **Resource Utilization Rate** μ thus is found to be

$$\begin{aligned} \mu &= P_{av} / P \\ &= (\sum_{i=1}^k s_i) / [k \cdot \max \{s_k : 1 \leq k \leq n\}]. \end{aligned} \quad (2.72)$$

When s_i for all i approaches to $\max \{s_k : 1 \leq k \leq n\}$, $\sum_{i=1}^k s_i = k \cdot \max \{s_k : 1 \leq k \leq n\}$, and consequently, μ approaches 1.

2.5.3 Resource Unlimited Speed-up and Utilization Rate

Suppose the number of resources $\geq n$, the number of program clauses. Then the concurrent resolution of different sets of clauses may take place in parallel. Suppose, out of s_1, s_2, \dots, s_k number of concurrent sets of resolvable clauses, r -sets of clauses on an average can participate in concurrent resolutions at the same time. Then the average time T_{RU} required to execute the program = $O[p(n - \sum_{i=1}^k s_i + k/r)]$. Then, **Resource Unlimited Speed-up**

$$\begin{aligned} S_{RU} &= T_u / T_{RU} \\ &= (p \cdot n) / [p(n - \sum_{i=1}^k s_i + k/r)] \\ &= n / (n - \sum_{i=1}^k s_i + k/r) \end{aligned} \quad (2.73)$$

Consequently, maximum speed-up occurs when $\sum_{i=1}^k s_i$ approaches n , and the result is

$$(S_{RU})_{\max} = (n/k)r. \quad (2.74)$$

Further, maximum degree of parallelism in a resource-unlimited system is $P_{RU} = \sum_{i=1}^k s_i$, and the average degree of parallelism $P_{av} = \sum_{i=1}^r s_i$. Thus **Resource Utilization Rate** is given by

$$\begin{aligned} \mu &= P_{av} / P_{RU} \\ &= (\sum_{i=1}^r s_i) / (\sum_{i=1}^k s_i) \end{aligned} \quad (2.75)$$

In a special case, when $s_i = s$ for all $i = 1$ to k , the above ratio reduces to (r/k) .

It is to be noted that when more than one group of concurrently resolvable clauses participate in the resolution process at the same time, $\max\{s_k: 1 \leq k \leq n\}$ assumes the maximum of the sum of the concurrently resolvable group of clauses.

2.6 Conclusions

This chapter presented a new algorithm for automated reasoning in a logic program using extended Petri net models. Because of the structural advantage of Petri net models, the proposed algorithm is capable of handling AND-, OR-, and stream-parallelisms in a logic program. A complexity analysis of a logic program with n number of clauses and k sets of concurrently resolvable clauses reveals that the maximum speed-up factor of the proposed algorithm in the worst case is $O(n/k)$. Under no constraints on resources, the speed-up factor is improved further by an additional factor of r , where r denotes an average number of the concurrent sets of resolvable clauses. In the absence of any constraints to resources, the maximum resource utilization rate for the proposed algorithm having $s_1 = s_2 = \dots = s_n$ is $O(r/k)$. With limited resource architecture, the proposed algorithm can execute safely at the cost of extra computational time. The selection of dimensions of such limited resource architecture depends greatly on the choice of r . Selection of r in typical logic programs in turn may be accomplished by running Monte Carlo simulations for a large set of programs. A complete study of this, however, is beyond the scope of this book.

With the increasing use of logic programs in data modeling, its utility in the next generation commercial database systems will also increase in pace. Such systems require a specialized engine that supports massive parallelisms. The proposed computational model, being capable of handling all possible parallelisms in a logic program, is an ideal choice for exploration in commercial database systems. To meet this demand, a hardware realization of the proposed algorithm is needed. We hope for the future when a specialized MIMD engine for logic programming will be commercially available for applications in the next-generation database machines.

Exercises

1. Given below a set of clauses:

- i) $r \vee s \leftarrow p \wedge q.$
- ii) $R(Z, f(X)) \leftarrow P(X, Y), Q(Y, Z).$
- iii) $\text{Above}(a, c) \leftarrow \text{Above}(a, b), \text{Above}(b, c).$
- iv) $N(f(Y), X) \leftarrow L(X, Y, Z), M(f(X), Z).$

- a) Identify the literals in the head part.
- b) Identify the literals in the body part.
- c) List the functions.
- d) List the arguments of the head literals.
- e) List the arguments of the body literals.

[**Answers:** a) The literals in the head part:

- i) r, s ;
- ii) $R(Z, f(X))$;
- iii) $Above(a, c)$;
- iv) $N(f(Y), X)$.

b) The literals in the body part:

- i) p, q ;
- ii) $P(X, Y), Q(Y, Z)$;
- iii) $Above(a, b), Above(b, c)$;
- iv) $L(X, Y, Z), M(f(X), Z)$.

c) The functions:

- i) Nil ;
- ii) $f(X)$;
- iii) Nil ;
- iv) $f(Y), f(X)$.

d) The arguments of the head literals:

- i) Nil ;
- ii) $Z, f(X)$;
- iii) a, c ;
- iv) $f(Y), X$.

e) The arguments in the body literals:

- i) Nil ;
- ii) $X, Y \text{ \& } Y, Z$;
- iii) $a, b \text{ \& } b, c$;
- iv) $(X, Y, Z) \text{ \& } f(X), Z$.]

2. Identify the following from the given set of clauses

- i) ground literal,
- ii) goal clause or query,
- iii) fact,
- iv) Horn clause,
- v) non-horn clause,
- vi) propositional clause.

- a) $\text{Boy}(X) \leftarrow \text{Male-child}(X), \text{Non-adult}(X).$
- b) $\text{Boy}(X), \text{Girl}(X) \leftarrow \text{Male-child}(X), \text{Non-adult}(X).$
- c) $S \leftarrow P, Q, R.$
- d) $\text{Boy}(X) \leftarrow.$
- e) $\text{Boy}(\text{ram}) \leftarrow.$
- f) $\leftarrow \text{Boy}(Y).$

[Answers:

- i) Ground literal: $\text{Boy}(\text{ram})$
- ii) Goal clause or query: $\leftarrow \text{Boy}(Y).$
- iii) Fact: $\text{Boy}(\text{ram}) \leftarrow.$
- iv) Horn clause: $\text{Boy}(X) \leftarrow \text{Male-child}(X), \text{Non-adult}(X).$
 $S \leftarrow P, Q, R.$
 $\text{Boy}(X) \leftarrow.$
 $\text{Boy}(\text{ram}) \leftarrow.$
 $\leftarrow \text{Boy}(Y).$
- v) Non_Horn clause: $\text{Boy}(X), \text{Girl}(X) \leftarrow \text{Male-child}(X), \text{Non-adult}(X).$
- vi) Propositional clause: $S \leftarrow P, Q, R.]$

3. Translate the following clauses into English:

$\text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y) \wedge \text{Wife}(Z, X).$
 $\text{Mother}(s, l) \leftarrow.$
 $\text{Wife}(s, r) \leftarrow.$
 Query: $\leftarrow \text{Father}(r, l).$

[Hints: If X is the father of Y and Z is the wife of X, then Z is the mother of Y.
 s is the mother of l.
 s is the wife of r.
 Whether r is the father of l?]

4. Construct an Extended Logic Program from the following statements:

The animals that eat plants are herbivorous, the animals that eat animals are carnivorous, and the animals that eat plants and animals are omnivorous.

[Hints: $\text{Herbivorous}(X) \leftarrow \text{Animals}(X) \wedge \text{Plant}(Y) \wedge \text{Eats}(X, Y).$
 $\text{Carnivorous}(X) \leftarrow \text{Animals}(X) \wedge \text{Plant}(Z) \wedge \text{Eats}(X, Z).$
 $\text{Omnivorous}(X) \leftarrow \text{Animals}(X) \wedge \text{Plant}(Y) \wedge \text{Animals}(Z) \wedge \text{Eats}(X, Y) \wedge \text{Eats}(X, Z).]$

5. Given an expression

$$w = P(X, f(Y, Z), d)$$

and three substitution sets

$$\begin{aligned}s_1 &= \{a/X, b/Y, c/Z\} \\ s_2 &= \{g(Y)/X\} \\ s_3 &= \{g(W)/X, b/Y, c/Z\}.\end{aligned}$$

Evaluate ws_1 , ws_2 , and ws_3 .

[**Answers:** $ws_1 = P(a, f(b, c), d)$,
 $ws_2 = P(g(Y), f(Y, Z), d)$, and
 $ws_3 = P(g(W), f(b, c), d)$.]

6. Let w be an expression and ws be an expression after substitution what is the substitution set?

$$\begin{aligned}w: M(Z, Y) &\leftarrow F(X, Y), K(Z, X). \\ ws: M(s, j) &\leftarrow F(r, j), K(s, r).\end{aligned}$$

[**Answers:** $s: \{r/X, j/Y, s/Z\}$.]

7. Let s_1 and s_2 be two substitutions such that

$$\begin{aligned}s_1: &\{r/X\}, \text{ and} \\ s_2: &\{u(X)/Y\}.\end{aligned}$$

Evaluate the composition of the substitutions:

- $s_1 \Delta s_2$ and
- $s_2 \Delta s_1$.

[**Answers:** The composition of the substitutions:

$$\begin{aligned}\text{a) } s_1 \Delta s_2 &= \{r/X, u(X)/Y\}, \\ \text{b) } s_2 \Delta s_1 &= \{u(r)/Y, r/X\}.\end{aligned}$$

8. Verify the substitution set property 1, i.e., $(ws_1) \Delta s_2 = w(s_1 \Delta s_2)$ using the following items:

Let the expression $w: P(X, Y, Z)$

And the substitution sets $s_1: \{u(X)/Y, v(X)/Z\}$,

$$s_2: \{r/X\}.$$

[**Answers:** $(ws_1) \Delta s_2 = P(X, u(X), v(X))\{r/X\}$
 $= P(r, u(r), v(r))$.

$$\begin{aligned}w(s_1 \Delta s_2) &= (P(X, Y, Z))\{r/X, u(r)/Y, v(r)/Z\} \\ &= P(r, u(r), v(r)). \\ \therefore (ws_1) \Delta s_2 &= w(s_1 \Delta s_2).\end{aligned}$$

9. Verify the substitution set property 2, i.e., $(s_1 \Delta s_2) \Delta s_3 = s_1 \Delta (s_2 \Delta s_3)$ using the following items:

Let the substitution sets $s_1: \{r/X\}$,

$$s_2: \{u(X)/Y\}, \text{ and} \\ s_3: \{l/Z\}.$$

[Answers: $s_1 \Delta s_2 = \{r/X, u(X)/Y\}$
 $(s_1 \Delta s_2) \Delta s_3 = \{r/X, u(X)/Y, l/Z\}.$
 $s_2 \Delta s_3 = \{u(X)/Y, l/Z\}$
 $s_1 \Delta (s_2 \Delta s_3) = \{r/X, u(X)/Y, l/Z\}.$
 $\therefore (s_1 \Delta s_2) \Delta s_3 = s_1(s_2 s_3).]$

10. Verify the substitution set property 3, i.e., $s_1 \Delta s_2 \neq s_2 \Delta s_1$ using the following items:

Let the substitution sets $s_1: \{r/X\},$
and $s_2: \{u(X)/Y\}.$

[Answers: $s_1 \Delta s_2 = \{r/X, u(X)/Y\}$ and
 $s_2 \Delta s_1 = \{u(r)/Y, r/X\}.$
 $\therefore s_1 \Delta s_2 \neq s_2 \Delta s_1.]$

11. Which of the following clauses are resolvable and what is the resolvent?

i) $Cl_1: R(Y, X) \leftarrow P(X, Y).$
ii) $Cl_2: P(r, n) \leftarrow .$
iii) $Cl_3: R(n, r) \leftarrow .$

[Hints: As the same predicate is present in the head and the body part of the clause number cl_2 and cl_1 respectively, they are resolvable. However, due to presence of the same predicate in the head part of the clauses, cl_1 and cl_3 , they are not resolvable.]

12. By definition 8, show that the following is a set of resolvable clauses.

i) $Cl_1: R(Z, X) \leftarrow P(X, Y), Q(Y, Z).$
ii) $Cl_2: P(r, s) \leftarrow .$
iii) $Cl_3: \neg R(l, r) \leftarrow .$
iv) $Cl_4: Q(s, l) \leftarrow .$

[Hints: $Cl_{12}: R(Z, r) \leftarrow Q(s, Z).$
 $Cl_{13}: \leftarrow P(r, Y), Q(Y, l).$
 $Cl_{14}: R(l, X) \leftarrow P(X, s).$

As each of the clauses is resolvable with at least one of the set of clauses, producing a resolvent, according to definition 2.9, it is a set of resolvable clauses.]

13. Identify the definite program clause with definite goal.

$Cl_1: R, S \leftarrow P, Q.$
 $Cl_2: R \leftarrow P, Q.$
 $Cl_3: \leftarrow P, Q.$

[Hints: According to definition 2.10, $Cl_2: R \leftarrow P, Q.$ is a definite clause containing one atom in its head and $Cl_3: \leftarrow P, Q.$ is a definite goal with empty consequent vide definition 2.12.]

14. Construct the resolution tree for linear selection,

$R \leftarrow P, Q.$
 $S \leftarrow R.$
 $T \leftarrow S.$
 $Q \leftarrow.$
 $P \leftarrow.$
Goal: $\leftarrow T.$

15. Determine whether the following are orderly or order independent clauses. If orderly, verify whether single or multiple sequence. Determine the various orders of resolution in the following set of clauses.

$Cl_1: R(Z, X) \leftarrow P(X, Y), Q(Y, Z).$
 $Cl_2: P(r, a) \leftarrow.$
 $Cl_3: Q(a, k) \leftarrow.$
 $Cl_4: \neg R(k, r) \leftarrow.$

[Hints: The clauses are to be selected pair-wise according to some definite order from the set of resolvable clauses. Otherwise they fail to generate a solution. As for example, cl_2, cl_3 or cl_3, cl_4 cannot be resolved. But, we can get results by resolving the clauses following some definite orders.

The multiple orders are:

Sequence 1: Order 1: 1-2-3-4/ 2-1-3-4

$Cl_{12-3-4}: \emptyset$

Sequence 2: Order 2: 1-2-4-3/ 2-1-4-3

$Cl_{12-4-3}: \emptyset$

Sequence 3: Order 3: 1-3-2-4/ 3-1-2-4

$Cl_{13-2-4}: \emptyset$

Sequence 4: Order 4: 1-3-4-2/ 3-1-4-2

$Cl_{13-4-2}: \emptyset$

Sequence 5: Order 5: 1-4-2-3/ 4-1-2-3

$Cl_{14-2-3}: \emptyset$

Sequence 6: Order 6: 1-4-3-2/ 4-1-3-2

$$Cl_{14.3.2}: \emptyset]$$

16. Determine the final resolvent from the given set of clauses. If not possible, indicate why.

$Cl_1: \text{Son}(Y, X) \leftarrow \text{Father}(X, Y).$
 $Cl_2: \text{Mother}(i, a) \leftarrow \text{Son}(a, n), \text{Husband}(n, i).$
 $Cl_3: \text{Father}(Z, Y) \leftarrow \text{Mother}(X, Y), \text{Wife}(X, Z).$

[**Hints:** $Cl_{12}: \text{Mother}(i, a) \leftarrow \text{Father}(n, a), \text{Husband}(n, i).$
 $Cl_{12.3}$ is not possible as double resolution takes place.]

17. Show that the following clauses are order independent. Explain the reason for nonvalidity.

$Cl_1: \text{Boy}(X), \text{Girl}(X) \leftarrow \text{Child}(X).$
 $Cl_2: \text{Likes_to_play_indoor}(X) \leftarrow \text{Boy}(X), \text{Introvert}(X).$
 $Cl_3: \text{Likes_to_play_doll}(X) \leftarrow \text{Girl}(X), \text{Likes_to_play_indoor}(X),$
 $\text{Has_doll}(X).$

[**Hints:** Each clause is resolvable with any other clause.
 Nonvalidity: Ultimately, double resolution takes place.]

18. Show whether the following set of clauses is concurrently resolvable:

$Cl_1: R(Z, X) \leftarrow P(X, Y), Q(Y, Z).$
 $Cl_2: P(r, a) \leftarrow.$
 $Cl_3: Q(a, k) \leftarrow.$
 $Cl_4: \neg R(k, r) \leftarrow.$

[**Hints:** As the instantiated resolvent generated following multiple sequences yields a unique result, the set includes concurrently resolvable clauses.]

19. Test whether the following clauses are concurrently resolvable:

- a) $Cl_1: \text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y), \text{Son}(Y, Z), \text{Married_to}(X, Z).$
 $Cl_2: \text{Father}(r, l) \leftarrow \text{Has_one_son}(r, l), \text{Male}(r).$
 $Cl_3: \text{Female}(Z) \leftarrow \text{Mother}(Z, Y).$
 $Cl_4: \text{Son}(l, s) \leftarrow.$
- b) $Cl_1: \text{Mother}(Z, Y) \leftarrow \text{Father}(X, Y), \text{Son}(Y, Z), \text{Married_to}(X, Z).$
 $Cl_2: \text{Father}(r, l) \leftarrow \text{Has_one_son}(r, l), \text{Male}(r).$
 $Cl_3: \text{Female}(Z) \leftarrow \text{Mother}(Z, Y).$
 $Cl_4: \text{Son}(k, s) \leftarrow.$

[**Answers:** (a) Concurrently resolvable, (b) not concurrently resolvable].

20. Indicate a) the AND-parallel clauses and b) the OR-parallel clauses to concurrently resolve the rule cl_1 with the rest of the clauses cl_2 through cl_5 .

Cl_1 : Likes_to_play(X, Y) \leftarrow Child(X), Game(Y).
 Cl_2 : Child(r) \leftarrow .
 Cl_3 : Child(t) \leftarrow .
 Cl_4 : Game(c) \leftarrow .
 Cl_5 : Game(l) \leftarrow .

[**Answers:** a) The AND-parallel clauses:

- i) Cl_1, Cl_2, Cl_4 ,
- ii) Cl_1, Cl_2, Cl_5 ,
- iii) Cl_1, Cl_3, Cl_4 ,
- iv) Cl_1, Cl_3, Cl_5 .

b) The OR-parallel clauses:

- i) Cl_1, Cl_2 and Cl_1, Cl_3 ,
- ii) Cl_1, Cl_4 and Cl_1, Cl_5 .]

21. a) Verify whether concurrent resolution is valid for the following clauses:

Cl_1 : Game(Y) \leftarrow Child(X), Likes_to_play(X, Y).
 Cl_2 : Outdoor_game(Y), Indoor_game(Y) \leftarrow Game(Y).
 Cl_3 : Child(X) \leftarrow Boy(X).
 Cl_4 : Child(X) \leftarrow Girl(X).
 Cl_5 : Boy(X) \leftarrow Likes_to_play(X, Y), Outdoor_game(Y).
 Cl_6 : Girl(X) \leftarrow Likes_to_play(X, Y), Indoor_game(Y).
 Cl_7 : Likes_to_play(t, c) \leftarrow .
 Cl_8 : Likes_to_play(r, l) \leftarrow .
 Cl_9 : Outdoor_game(c) \leftarrow .
 Cl_{10} : \neg Indoor_game(c) \leftarrow .

- b) If yes, identify the types of concurrent resolution for the following sequences:

- i) Cl_5, Cl_7, Cl_9 ,
- ii) Cl_1, Cl_2, Cl_5 ,
- iii) Cl_3, Cl_5, Cl_7, Cl_9 ,
- iv) Cl_1, Cl_2, Cl_6 ,
- v) Cl_1, Cl_2, Cl_3, Cl_7 .

[**Hints:**

a) For the following ordered sequence of clauses for SLD resolution, the final resolvent is identical. So, concurrent resolution is possible in the following cases:

- i) **Sequence 1:** Order 1: 1-3-7.
Sequence 2: Order 2: 1-7-3.
- ii) **Sequence 3:** Order 3: 1-3-8.
Sequence 4: Order 4: 1-8-3.
- iii) **Sequence 5:** Order 5: 1-4-7.
Sequence 6: Order 6: 1-7-4.
- iv) **Sequence 7:** Order 7: 1-4-8.
Sequence 8: Order 8: 1-8-4.
- v) **Sequence 9:** Order 9: 5-8-9.
Sequence 10: Order 10: 5-9-8.

b)

- i) Concurrent resolution between a rule and facts,
- ii) Concurrent resolution between rules,
- iii) Concurrent resolution between rules and facts,
- iv) Concurrent resolution between rules,
- v) Concurrent resolution between fact and rules.]

22. Map the following FOL clauses on to an EPN and state the result after resolution.

Cl_1 : $\text{Game}(Y) \leftarrow \text{Child}(X), \text{Likes_to_play}(X, Y).$
 Cl_2 : $\text{Outdoor_game}(Y), \text{Indoor_game}(Y) \leftarrow \text{Game}(Y).$
 Cl_3 : $\text{Child}(t) \leftarrow.$
 Cl_4 : $\text{Likes_to_play}(t, c) \leftarrow.$
 Cl_5 : $\neg \text{Indoor_game}(c) \leftarrow.$

23. Map the following program clauses onto an EPN. What result do you obtain after execution of the program by the algorithm: 'Procedure Automated Reasoning'?

The program clauses are:

Cl_1 : $\text{Reproduce_by_laying_eggs}(X) \leftarrow \text{Build_nests}(X), \text{Lay_eggs}(X).$
 Cl_2 : $\text{Has_wings}(X) \leftarrow \text{Can_fly}(X), \text{Has_feather}(X).$
 Cl_3 : $\text{Bird}(X) \leftarrow \text{Reproduce_by_laying_eggs}(X), \text{Has_beaks}(X), \text{Has_wings}(X).$
 Cl_4 : $\text{Build_nests}(p) \leftarrow.$
 Cl_5 : $\text{Lay_eggs}(p) \leftarrow.$
 Cl_6 : $\text{Can_fly}(p) \leftarrow.$
 Cl_7 : $\text{Has_feather}(p) \leftarrow.$
 Cl_8 : $\text{Has_beaks}(p) \leftarrow.$

24. Given a set of clauses (Cl_1 - Cl_{13}):

- Cl_1 : Father(Y, Z), Uncle(Y, Z) \leftarrow Father(X, Y), Grandfather(X, Z).
 Cl_2 : Paternal_uncle(X, Y), Maternal_uncle(X, Y) \leftarrow Uncle(X, Y).
 Cl_3 : Mother(Z, Y) \leftarrow Father(X, Y), Married_to(X, Z).
 Cl_4 : Father(X, Y) \leftarrow Mother(Z, Y), Married_to(X, Z).
 Cl_5 : Father <r, n> \leftarrow .
 Cl_6 : Father <r, d> \leftarrow .
 Cl_7 : \neg Father <d, a> \leftarrow .
 Cl_8 : Grandfather <r, a> \leftarrow .
 Cl_9 : \neg Paternal_uncle <n, a> \leftarrow .
 Cl_{10} : \neg Maternal_uncle <n, a> \leftarrow .
 Cl_{11} : \neg Maternal_uncle <d, a> \leftarrow .
 Cl_{12} : Married_to <r, t> \leftarrow .
 Cl_{13} : Married_to <n, i> \leftarrow .

- List the possible resolutions that take place in the network.
- Identify the concurrent resolutions among those in the above list.
- Represent the concurrent resolutions in tabular form like Table 2.1.

[Hints: a) The possible resolutions are:

1-3, 1-4, 1-5, 1-6, 1-7, 1-8, 2-9, 2-10, 2-11, 3-12, 3-13, 4-12, 4-13.

b) The concurrent resolutions are:

Sequence 1: Order 1: 1-6-7-8.

$Cl_{1-6-7-8} = \text{Uncle}(d, a) \leftarrow$.

Sequence 2: Order 2: 2-9-10.

$Cl_{2-9-10} = \leftarrow \text{Uncle}(n, a) \equiv \neg \text{Uncle}(n, a) \leftarrow$.

Sequence 3: Order 3: 3-5-12.

$Cl_{3-5-12} = \text{Mother}(t, n) \leftarrow$.

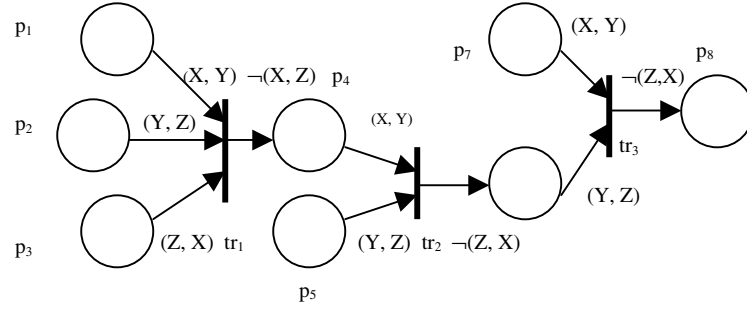
Sequence 4: Order 4: 3-6-12.

$Cl_{3-6-12} = \text{Mother}(t, d) \leftarrow$.

- The table describing concurrent resolution for problem 24 is given below.

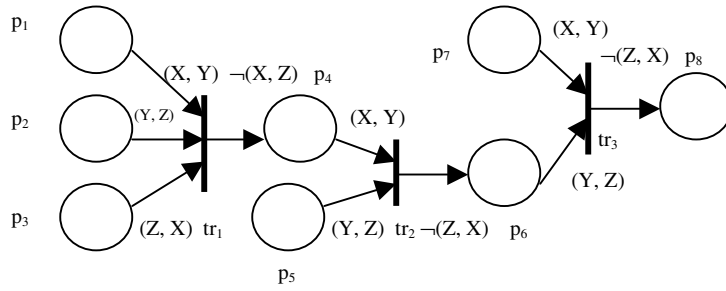
Time slot	Transitions	Set of c-b	Set of u-b	Flag=0, if c-b \notin u-b = 1, if c-b \in u-b $\neq \{\emptyset\}$ or c-b = $\{\emptyset\}$
First cycle	tr ₁	{r/X, d/Y, a/Z}	{ $\{\emptyset\}$ }	0
	tr ₂	{n/X, a/Y}	{ $\{\emptyset\}$ }	0
	tr ₃	{r/X, n/Y, t/Z}/ {r/X, d/Y, t/Z}	{ $\{\emptyset\}$ }	0
Second cycle	tr ₁	{r/X, n/Y, a/Z}	{ {r/X, d/Y, a/Z} }	0
	tr ₂	{d/X, a/Y}	{ {n/X, a/Y} }	0
	tr ₃	{ \emptyset }	{ {r/X, n/Y, t/Z}, {r/X, d/Y, t/Z} }	0
Third cycle	tr ₁	{ \emptyset }	{ {r/X, d/Y, a/Z} }	1
	tr ₂	{ \emptyset }	{ {n/X, a/Y} }	1
	tr ₃	{n/X, a/Y, i/Z}	{ {r/X, n/Y, t/Z}, {r/X, d/Y, t/Z}, {n/X, a/Y, i/Z} }	1

- 25.a) For answering the goal($\leftarrow M(t, n).$) with the clauses given in problem 24, draw the SLD-tree from the given set of clauses.
- b) Use 'Procedure Automated_Reasoning' to verify the goal.
- c) Assuming unit time to perform a resolution, determine the computational time involved for execution of the program by SLD-tree approach.
- d) Compute the computational time required for execution of the program clauses on EPN using 'Procedure Automated_Reasoning'.
- e) Determine the percentage time saved in the EPN approach $((a-b)/a \times 100)$ where 'a' stands for SLD, 'b' for EPN.
26. From the given Petri nets (Fig. 2.8 and Fig. 2.9), calculate the speed-up and the Resource utilization rate from the definition given in the equations (2.68) and (2.72).



Token at the place $p_1 = \langle a, b \rangle$, Token at the place $p_2 = \langle b, c \rangle$, Token at the place $p_3 = \langle c, a \rangle$, Token at the place $p_5 = \langle c, d \rangle$, Token at the place $p_7 = \langle e, d \rangle$.

Fig. 2.8: A Petri net.



Token at the place $p_1 = \langle a, b \rangle$, Token at the place $p_2 = \langle b, c \rangle$, Token at the place $p_3 = \langle c, a \rangle$, Token at the place $p_5 = \langle c, d \rangle$, Token at the place $p_7 = \langle e, d \rangle$, Token at the place $p_8 = \neg \langle a, e \rangle$.

Fig. 2.9: An illustrative Petri net.

[Hints: Pipelining of transitions in the first and the second cases are given in Figures 2.10 and 2.11 respectively.

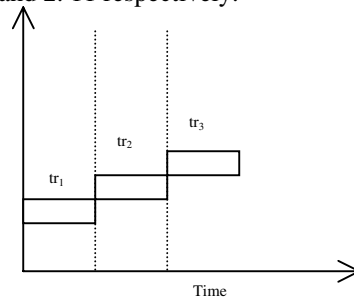


Fig. 2.10: Diagram showing pipelining of transitions for the first case.

Here, in the case of the first Petri net, as per definition:

$$n = 8,$$

$$\sum s_i = s_1 + s_2 + s_3 = 4+2+2 = 8,$$

$$\sum m_i = \sum s_i + \sum d_i = 8+3 = 11,$$

$$k = 3$$

$$\therefore \text{Speed-up factor } S = T_u/T_m = n / (n - \sum s_i + k) = 8/3 = 2.66.$$

$$\begin{aligned} \& \text{ Resource utilization rate } \mu = \sum s_i / [k \cdot \max \{s_k: 1 \leq k \leq n\}] \\ &= 8/(3 \times 4) = 8/12 = 0.66. \end{aligned}$$

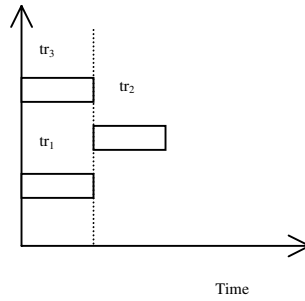


Fig. 2.11: Diagram showing pipelining of transitions for the second case.

Now, for the second case, $n = 9$,

$$\sum s_i = 4+2+3 = 9,$$

$$k = 2.$$

$$\therefore \text{The speed-up factor } S = T_u/T_m = n / (n - \sum s_i + k)$$

$$= 9/(9 - \sum s_i + k) = 9/2 = 4.5.$$

$$\& \text{ Resource utilization rate } \mu = \sum s_i / [k \cdot \max \{s_k: 1 \leq k \leq n\}]$$

$$= 9/(2 \times 7) = 9/14 = 0.64.]$$

References

- [1] Bhattacharya, A., Konar, A. and Mandal, A. K., "A parallel and distributed computational approach to logic programming," Proc. of *International Workshop on Distributed Computing (IWDC 2001)*, held in Calcutta, December 2001.

- [2] Ganguly, S., Silberschatz, A., Tsur, S., "Mapping datalog program execution to networks of processors," *IEEE Trans. on Knowledge and Data Engg.*, vol. 7, no. 3, June 1995.
- [3] Hermenegildo, M. and Tick, E., "Memory performance of AND-parallel PROLOG on shared-memory architecture," *Proc. of the 1988 Int. Conf., on Parallel Processing*, vol. II, Software, pp. 17-21, Aug. 15-19, 1988.
- [4] Hwang, K. and Briggs, F. A., *Computer Architecture and Parallel Processing*, McGraw-Hill, pp. 27-35, 1986.
- [5] Jeffrey, J., Lobo, J. and Murata, T., "A high-level Petri net for goal-directed semantics of Horn clause logic," *IEEE Trans. on Knowledge and Data Engineering*, vol. 8, no. 2, April 1996.
- [6] Kale, M. V., Parallel Problem Solving, In *Parallel Algorithms for Machine Intelligence and Vision*, Kumar, V., Gopalakrishnan, P. S. and Kanal, L.N. (Eds.), Springer-Verlag, Heidelberg, 1990.
- [7] Konar, A., *Uncertainty Management in Expert Systems Using Fuzzy Petri Nets*, Ph.D. Thesis, Jadavpur University, 1994.
- [8] Konar, A. and Mandal, A. K., "Uncertainty management in expert systems using fuzzy Petri nets," *IEEE Trans. on Knowledge and Data Engg.*, vol. 8., no.1, Feb. 1996.
- [9] Li, L., "High level Petri net model of logic program with negation," *IEEE Trans. on Knowledge and Data Engg.*, vol. 6, no. 3, June 1994.
- [10] Murata, T. and Yamaguchi, H., "A Petri net with negative tokens and its application to automated reasoning," *Proc. of the 33rd Midwest Symp. on Circuits and Systems*, Calgary, Canada, Aug. 12-15, 1990.
- [11] Naganuma, J., Ogura, T., Yamada, S-I., and Kimura, T., "High-speed CAM-based Architecture for a Prolog Machine (ASCA)," *IEEE Transactions on Computers*, vol. 37, no.11, November 1988.
- [12] Patt, Y. N., "Alternative Implementations of Prolog: the micro architecture perspectives," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, no.4, July/August 1989.
- [13] Peterka, G. and Murata, T., "Proof procedure and answer extraction in Petri net model of logic programs," *IEEE Trans. on Software Eng.*, vol. 15, no. 2, Feb. 1989.

- [14] Takeuchi, A., *Parallel Logic Programming*, Wiley, 1992.
- [15] Yan, J. C., "Towards parallel knowledge processing," In *Advanced series on Artificial Intelligence: Knowledge Engineering Shells, Systems and Techniques*, Bourbakis, N. G. (Ed.), vol. 2, World Scientific, 1993.

Cognitive Engineering

A Distributed Approach to Machine Intelligence

Konar, A.

2005, XVIII, 354 p. 124 illus., Hardcover

ISBN: 978-1-85233-975-3