5. If S and T are extensions of I and $S \subseteq T$, then $S = T$.

6. Prove that the minimal Herbrand models of programs P and Q of Example 7.10 are the sets M_1 and M_2 given there.

7. Prove that M_1 is the only stable model of P in Example 7.10.

   (Hint: To begin the analysis note that any candidate must contain $p(1,2)$ to be a model of P but will not contain any other instance of P by minimality considerations.)

Refer to Exercises I.6.7–8 for the basic terminology about graphs and partial orderings used below.

8. Let n be a natural number and G be a locally finite graph, i.e., a graph in which, for each node x there are at most finitely many nodes y such that $\{x, y\}$ is an edge of G. We define a nonmonotonic formal system $\langle U(G), N(G) \rangle$ by first setting $U(G) = \{Cxi | \ x \text{ is a node of } G \text{ and } i \leq n\}$. We then put into $N(G)$, for each node x of G and $j \leq n$, the rule

$$\frac{: Cx1, \ldots, Cx(j-1), Cx(j+1), \ldots, Cxn}{Cxj}.$$

   Finally, we put into $N(G)$, for each pair x, y of distinct nodes of G, each $i \leq n$ and each $\varphi \in U(G)$, the rule

$$\frac{Cxi, Cyi}{\varphi}.$$

   Prove that $S \subseteq U(G)$ is an extension for $\langle U(G), N(G) \rangle$ if and only if coloring each node x of G with color i iff $Cxi \in S$ produces an n–coloring of G.

9. Let P be a partial ordering of width n. We define a nonmonotonic system $\langle U(P), N(P) \rangle$ by first setting $U(P) = \{Cxi | x \in P \text{ and } i \leq n\}$. For each $x \in P$ we put into $N(P)$ the rule

$$\frac{: Cx1, \ldots, Cx(j-1), Cx(j+1), \ldots, Cxn}{Cxj}.$$

   Finally, for each x and y in P that are incomparable in the partial ordering, we put into $N(P)$ the rule

$$\frac{Cxi, Cyi}{\varphi}.$$

   Prove that $S \subseteq U$ is an extension of $\langle U(P), N(P) \rangle$ if and only if the set $\{C_1, \ldots, C_n\}$ is a collection of disjoint chains covering P where $C_i = \{x | C_{x_i} \in S\}$.

8 Computability and Undecidability

One of the major tasks of the logicians of the 30s and 40s was the formalization of the basic intuitive notion of an algorithm or an effective procedure. (For convenience we consider procedures on the natural numbers.) Many seemingly different definitions were proposed by a number of researchers including Church, Gödel, Herbrand, Kleene, Markov and Post. They suggested schemes involving recursion, equational deduction systems, idealized models of computing machines and others. Perhaps the philosophically most convincing proposal was that of Turing. He gave what is undoubtedly now the best known definition in terms of a simple machine model of computation: the Turing machine.

Every function calculable by any of these models was clearly effective. As investigations progressed, it became evident that any function that was intuitively computable could be calculated in any of the systems. Indeed, over a number of years all these proposals were proven equivalent, that is, the class of functions computable in any one model is the same as that computable in any other. These functions are now called the *recursive functions*. Early results along these lines led Church to formulate what is now known as *Church's thesis*: the effectively calculable functions are precisely the recursive ones. The weight of the evidence has by now produced an almost universal acceptance of this thesis. Thus, to prove that any computation scheme is universal in the sense that it computes every effective function, it suffices to prove that it computes every function computable by any of the schemes known to define the class of recursive functions.

It is not difficult to model almost any of the standard definitions by deduction in predicate logic: for each recursive function f, we can write down axioms in a language for arithmetic which includes a term $\bar{n}$ for each natural number n and a two–place predicate symbol p_f such that $f(n) = m$ iff $p_f(\bar{n}, \bar{m})$ is a logical consequence of the axioms. (We restrict our attention to unary functions simply to avoid strings of variables. Everything we do will work just as well for functions of any specified arity m by simply replacing the single variable x_1 by a sequence of variables $x_1, x_2, \ldots, x_m$.) For the most part, these representations can be naturally expressed in the form of PROLOG programs. (See Exercises 1–2 for an example.) Thus, any sound and complete implementation of PROLOG (e.g., with breadth–first searching) will correctly compute all recursive functions. By choosing the right model of computation (Shepherdson's register machines as described in Definition 8.1) and exercising some cleverness in the translation into PROLOG (Definition 8.4), we prove that the standard implementation via the leftmost literal selection rule and depth-first searching of the SLD–tree also suffices to compute all recursive functions. (In fact, the "right" programs will run correctly with essentially any selection rule and search procedure.) Thus PROLOG is a universal computing machine model (Corollaries 8.6–8.7).

Once one has an agreed–upon mathematical definition of an algorithm or the class of effectively computable functions, one can hope to prove that various procedures (or decisions) cannot be carried out (made) by any algorithm or that