The connection between pattern matching and constraint satisfaction is actually very close. We can view each conjunct as a constraint on the variables that it contains—for example, \( \text{Missile}(x) \) is a unary constraint on \( x \). Extending this idea, we can express every finite-domain CSP as a single definite clause together with some associated ground facts.

Consider the map-coloring problem from Figure 9.1, shown again in Figure 9.5(a). An equivalent formulation as a single definite clause is given in Figure 9.5(b). Clearly, the conclusion \( \text{Colorable()} \) can be inferred only if the CSP has a solution. Because CSPs in general include 3-SAT problems as special cases, we can conclude that matching a definite clause against a set of facts is NP-hard.

It might seem rather depressing that forward chaining has an NP-hard matching problem in its inner loop. There are three ways to cheer ourselves up:

- We can remind ourselves that most rules in real-world knowledge bases are small and simple (like the rules in our crime example) rather than large and complex (like the CSP formulation in Figure 9.5). It is common in the database world to assume that both the sizes of rules and the arities of predicates are bounded by a constant and to worry only about data complexity—that is, the complexity of inference as a function of the number of ground facts in the knowledge base. It is easy to show that the data complexity of forward chaining is polynomial.

- We can consider subclasses of rules for which matching is efficient. Essentially every Datalog clause can be viewed as defining a CSP, so matching will be tractable just when the corresponding CSP is tractable. Chapter 6 describes several tractable families of CSPs. For example, if the constraint graph (the graph whose nodes are variables and whose links are constraints) forms a tree, then the CSP can be solved in linear time. Exactly the same result holds for rule matching. For instance, if we remove South

---

**Figure 9.5**

(a) Constraint graph for coloring the map of Australia. (b) The map-coloring CSP expressed as a single definite clause. Each map region is represented as a variable whose value can be one of the constants \( \text{Red}, \text{Green} \) or \( \text{Blue} \).
Australia from the map in Figure 9.5, the resulting clause is

\[ Diff(\text{wa}, \text{nt}) \land Diff(\text{nt}, \text{aq}) \land Diff(\text{aq}, \text{nsw}) \land Diff(\text{nsw}, \text{v}) \land \text{Colorable()} \]

which corresponds to the reduced CSP shown in Figure 6.12 on page 224. Algorithms for solving tree-structured CSPs can be applied directly to the problem of rule matching.

- We can try to eliminate redundant rule-matching attempts in the forward-chaining algorithm, as described next.

Incremental forward chaining

When we showed how forward chaining works on the crime example, we cheated; in particular, we omitted some of the rule matching done by the algorithm shown in Figure 9.3. For example, on the second iteration, the rule

\[ \text{Missile}(x) \land \text{Weapon}(x) \]

matches against \( \text{Missile}(\text{Mi}) \) (again), and of course the conclusion \( \text{Weapon}(\text{Mi}) \) is already known so nothing happens. Such redundant rule matching can be avoided if we make the following observation: Every new fact inferred on iteration \( t \) must be derived from at least one new fact inferred on iteration \( t - 1 \). This is true because any inference that does not require a new fact from iteration \( t - 1 \) could have been done at iteration \( t - 1 \) already.

This observation leads naturally to an incremental forward-chaining algorithm where, at iteration \( I \), we check a rule only if its premise includes a conjunct \( p_i \) that unifies with a fact \( p_i \) newly inferred at iteration \( t - 1 \). The rule-matching step then fixes \( p_i \) to match with \( p_i' \) but allows the other conjuncts of the rule to match with facts from any previous iteration. This algorithm generates exactly the same facts at each iteration as the algorithm in Figure 9.3, but is much more efficient.

With suitable indexing, it is easy to identify all the rules that can be triggered by any given fact, and indeed many real systems operate in an "update" mode wherein forward chaining occurs in response to each new fact that is TELLed to the system. Inferences cascade through the set of rules until the fixed point is reached, and then the process begins again for the next new fact.

Typically, only a small fraction of the rules in the knowledge base are actually triggered by the addition of a given fact. This means that a great deal of redundant work is done in repeatedly constructing partial matches that have some unsatisfied premises. Our crime example is rather too small to show this effectively, but notice that a partial match is constructed on the first iteration between the rule

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \land \text{Criminal}(x) \]

and the fact \( \text{American}(\text{West}) \). This partial match is then discarded and rebuilt on the second iteration (when the rule succeeds). It would be better to retain and gradually complete the partial matches as new facts arrive, rather than discarding them.

The \texttt{rete} algorithm\(^1\) was the first to address this problem. The algorithm preprocesses the set of rules in the knowledge base to construct a sort of dataflow network in which each

\(^1\text{Rete is Latin for net. The English pronunciation rhymes with treaty.}\)