Notes on Satisfiability-Based Problem Solving

Representing Problems: Examples from the 2015 LP/CP Programming Competition

David Mitchell
mitchell@cs.sfu.ca

February 8, 2020

Preliminary draft. Please do not distribute. Corrections and suggestions welcome.

1 Introduction

The following five problems are variants of the problems from the 2015 LP/CP Programming Competition. The original versions, together with some sample instances, can be found on the competition web page, at `http://picat-lang.org/lp_cp_pc/`.

1.1 Conventions and Abbreviations

It is often convenient to let the universe (or part of the universe) of a structure be some prefix of the natural numbers (or the natural numbers plus 0). We will do this frequently in the following notes. When this is the case, we will often assume that our vocabulary includes the standard order $<$ on the natural numbers. When the universe is $[n] = \{1, 2, \ldots, n\}$, we often assume our vocabulary has the constant symbols 1 and max, which will always be interpreted as 1 and n, and sometimes addition or subtraction of 1.

We will often use the notation $P(x, y)$ for formula Pxy, so that we can use mnemonic variable names without introducing parsing ambiguity. Because some formulas will have many parentheses, we will often use [and] for parentheses to improve readability.

We will find a number of additional formula abbreviations useful:

Abbreviation	Formula
$\forall x \neq y\ \phi$	$\forall x(x \neq y \rightarrow \phi)$
$\exists x \neq y\ \phi$	$\exists x(x \neq y \wedge \phi)$
$\exists! x\ \phi(x)$	$\exists x(\phi(x) \wedge \forall y \neq x\ \neg\phi(y))$
$\forall x < y\ \phi$	$\forall x(x < y \rightarrow \phi)$

$\exists x < y\ \phi$	$\exists x(x < y \wedge \phi)$	
$x < y < z$	$(x < y \wedge y < z)$	
$x \leq y$	$(x < y \vee x = y)$	
$x > y$	$y < x$	
$x \in [n]$	$(1 \leq x \leq n)$	where n is a constant symbol (e.g., *max*)
$\forall x \in [n]\ \phi$	$\forall x[(1 \leq x \leq n) \rightarrow \phi]$	where n is a constant symbol (e.g., *max*)
$\exists x \in [n]\ \phi$	$\exists x[(1 \leq x \leq n) \wedge \phi]$	where n is a constant symbol (e.g., *max*)
$\forall x \in S\ \phi$	$\forall x(Sx \rightarrow \phi)$	where S is a unary relation symbol
$\exists x \in S\ \phi$	$\exists x(Sx \wedge \phi)$	where S is a unary relation symbol

2 "Logistics" Problem: A Simple Use of Order

The original description is motivated in terms of trucks moving goods to locations on a map, a kind of logistics problem. The problem to be solved, though, is the following simple spanning-subgraph problem on edge-weighted undirected graphs.

Given: - Undirected graph $G = \langle V, E \rangle$, with positive edge weights,
- Distinguished $s \in V$, the "start vertex",
- Set $D \subset V$, of "destination vertices",
- Natural number $k > 0$, the "cost bound".

Find: A connected subgraph of G, with total weight at most k, that spans $D \cup \{s\}$.

Graph connectivity is not FO definable, so we will need to introduce a second order variable to help express the property. For this particular problem, some observations about optimal solutions allow us to simplify a bit from the general case of connectivity.

Proposition 1. *Let $G = \langle V, E \rangle$ be a graph with non-negative edge weights, and $S \subset V$. If G is has a connected subgraph of weight w that spans S, then G has a subgraph of weight at most w that spans S and is a tree.*

To see this, consider a sub-graph H that spans S and has weight w. If H has a cycle, remove one edge from the cycle. The resulting graph still spans S, and has weight at most w. Repeat until there are no cycles.

So, it is sufficient to search for a suitable tree in G. Let the tree be $T = \langle U, F \rangle$. We require that T have the following properties:

1) Every edge of T is an edge of E,

2) T spans $D \cup \{s\}$ (i.e., every vertex in $D \cup \{s\}$ belongs to some edge of T),
3) T is connected,
4) The sum of the weights of edges in F is at most k.

The first two properties are easy to specify. Connectivity requires some effort, but it is a bit simpler for trees than for arbitrary graphs, due to the following property.

Proposition 2. *A graph $G = \langle V, E \rangle$ is a tree iff there is an ordering on V such that every vertex, except the first vertex in the ordering, has exactly one neighbour earlier in the ordering.*

To see one direction, suppose we have such an ordering. Then there is a path from every vertex to the first, so G is connected. Also, there are exactly $|V| - 1$ edges so G is a tree. For the other direction, if G is a tree, we may construct a suitable order by choosing any vertex to be first, and enumerating the remaining vertices according to a pre-order traversal. The choice of root induces a direction on each edge (e.g., from child to parent). In the enumeration, all descendents of a vertex u appear after u. Therefore, the for each vertex u other than the root, the edge to its (unique) parent is the desired unique edge to a vertex earlier in the ordering.

Each problem instance will be a structure for vocabulary $[E, D, s, k]$ in where:

- The universe is the set $V = [n] = \{1, \dots n\}$ of vertices,
- The edge relation E is a ternary relation, where $E(u, v, w)$ means that there is an edge (u, v) with weight w,
- s is a constant symbol denoting the “start” vertex,
- D is a unary relation symbol denoting the set of “destination” vertices,
- k is a constant symbol denoting the maximum weight (cost) a solution may have.

Remark 1. By setting our universe to be $[n]$, we have restricted the edge weights, and also the weight bound k, so be in $[n]$. This is a bit un-natural, but done for simplicity.

A solution for an instance will be a tree $T = \langle U, F \rangle$, so our vocabulary for writing our specification formulas will have a unary relation symbol U and a ternary relation symbol F. We will also use the binary relation symbol $<$, for the standard order on the set $[n]$. We need to construct a new ordering on V to verify T is a tree using Proposition 2. For this, we use a unary function symbol p, which will denote a permutation on V. This permutation induces an order on V in which v precedes u iff $p(v) < p(u)$.

We have the following constraints on T and p to ensure that $T = \langle U, F \rangle$ is a solution, corresponding to the four properties identified above. (Constraints 3 and 4 enforce the connectivity property.)

1. $D \cup \{s\} \subset U$.
2. Every edge in F is an edge in E. (It is tempting to write $F \subset E$, but this may not be strictly so: See Assumption 1 below, and the following description of F.)
3. p is a permutation of U. (It is natural to read $p(x)$ as "the position of x in the ordering".)
4. For each vertex $v \in U$ except the first one in the order induced by p (the one with $p(v) = 1$), there is exactly one vertex $u \in U$ with $F(u, v, w)$ and $p(u) < p(v)$.
5. $(\sum_{\{u,v,w|F(u,v,w)\}} w) \leq k$.

Assumption 1. *We assume (as is typically the case in practice) that the actual ternary relation E we are given contains only one tuple for each edge. That is, if there is an edge (u, v) in G of weight w, then E contains either the tuple $\langle u, v, w\rangle$ or the tuple $\langle v, u, w\rangle$, but not both, even though G is undirected. We will have to take this into account in our formulas that use E.*

Edges in T will be directed (because it seems easier that way), and in particular $F(u, v, w)$ will mean that there is an arc from v to u, and that u is closer than v to the root of T (so u precedes v in the order induced by p).

We can now express our constraints with the following formulas.

1. $\forall v[\ (D(v) \vee v{=}s) \rightarrow U(v)\ ]$
2. $\forall u \forall v \forall w[\ F(u, v, w) \rightarrow (E(u, v, w) \vee E(v, u, w))\ ]$
3. $\forall x{\in}U\ \exists! y{\in}U\ (p(y) = x)$
4. $\forall v{\in}U[(p(v){>}1) \rightarrow \exists! u{\in}U\ (p(u){<}p(v)\ \wedge\ \exists w F(u, v, w))\ ]$
5. $sum(\langle u, v, w\rangle,\ F(u, v, w),\ w) \leq k$

Formula 5 involves a term that computes the sum of the weights of the edges in F. The first argument of the term is the tuple of variables we are considering values of, the second argument is a formula restricting the values to those we want to sum over (the edges of F with their weights) and the third is the term to be summed.

The general form of this term is $sum(\bar{x}, \phi(\bar{x}), t(\bar{x}))$, where $\bar{x}$ is a tuple of variables, $\phi(\bar{x})$ is a formula whose free variables are among those of $\bar{x}$, and t is an arithmetic term (that is, its value is a number) whose variables are among those of $\bar{x}$. The interpretation of this term in a structure is as follows: ϕ defines a set of tuples; t maps each tuple to some value, and the sum of all those values (which may not all be distinct) is the value of the term.

We could write an $\exists SO$ formula that defines the function *sum*. However, almost all practical solver languages have a similar summation function, or have arithmetic expressions that make it easily definable, so we will take it as given (along with $=$ and $<$).

Let Φ be the conjunction of the formulas 1 through 5. A solution for instance structure $\mathcal{A}$ for vocabulary $[E, D, s, k]$ is obtained by finding a structure $\mathcal{B}$ that is an expansion of $\mathcal{A}$ to the vocabulary $[E, D, s, k, U, F]$ and that satisfies Φ. The desired tree is $T = \langle U^{\mathcal{B}}, F^{\mathcal{B}} \rangle$, and the cost is the sum of weights of edges in $F^{\mathcal{B}}$.

3 Pizza Coupon Problem

The problem is posed as follows. We have a list of pizzas we want to buy, each with a price. (Here we care only about prices, not what sorts of pizzas they are.) We also have a collection of "buy x, get y free" coupons ("vouchers" in the original description), each of which can be used to get a specified number of pizzas free, provided we have paid for some other specified number of pizzas. Each pizza that we get free by applying a coupon c must have a price no more than that of the cheapest paid-for pizza used to justify using coupon c.

Our instance vocabulary will be $[price, buy, free, n, m]$, where *price*, *buy* and *free* are unary function symbols and n and m are constant symbols. An instance structure consists of:

- The number n of pizzas;
- The number m of coupons;
- Unary function $price : [n] \rightarrow \mathbb{N}$, giving the price for each of the n pizzas;
- Unary function $buy : [m] \rightarrow \mathbb{N}$, giving the number of paid pizzas required to justify using each of the m coupons;
- Unary function $free : [m] \rightarrow \mathbb{N}$, giving the number of free pizzas that can be obtained by using each of the m coupons;
- Cost bound $k \in \mathbb{N}$.

The universe consists of all the numbers appearing in the structure.

The goal is to find a selection of coupons, and "assignments" of pizzas to coupons (the paid pizzas and free pizzas associated with each coupon we use), that allows us to get all the desired pizzas for total cost at most k.

We will use the following vocabulary symbols (as well as the instance vocabulary):

- Unary relation symbol *Paid*, for the set of pizzas will be paid for;
- Unary relation symbol *Used*, for the set of coupons that will be used;
- Binary relation symbol *Justifies*, where $\mathit{Justifies}(c, p)$ holds if pizza p is one of the pizzas we will pay for to justify using coupon c;
- Binary relation symbol *UsedFor*, where $\mathit{UsedFor}(c, p)$ holds if p is one of the pizzas we get free by using coupon c.

In our solution, we will need to count the number of items in a set defined by a formula. For this purpose, we use cardinality (or counting) terms of the form

$$\#(\bar{x}, \phi(\bar{x}\bar{y})).$$

Given a structure $\mathcal{A}$ for the vocabulary of ϕ, and a valuation σ, $\#(\tilde{x}, \phi(\bar{x}\bar{y}))^{\mathcal{A}}[\sigma]$ is the number of distinct tuples $\bar{a} \in A^k$, where $k = |\bar{x}|$, such that $\mathcal{A} \models \phi[\sigma(\bar{x} \to \bar{a})]$.

To find a solution to instance $\mathcal{A}$, we need to find a structure $\mathcal{B}$ that is an expansion of $\mathcal{A}$ to the vocabulary $[\mathit{price}, \mathit{buy}, \mathit{free}, n, m, \mathit{Paid}, \mathit{Used}, \mathit{Justifies}, \mathit{UsedFor}]$ and that satisfies the following formulas.

1. We pay for exactly the pizzas that we don't get free by using coupons:

   $$\forall p[\mathit{Paid}(p) \leftrightarrow \neg\exists c\, \mathit{UsedFor}(c, p)]$$

   (Notice that, if *UsedFor* has the intended interpretation, then $\exists c\, \mathit{UsedFor}(c, p)$ defines the set of free pizzas.)

2. *Used* is the set of coupons that we use:

   $$\forall c[\mathit{Used}(c) \leftrightarrow \exists p\, \mathit{UsedFor}(c, p)]$$

   (Notice that, if *UsedFor* is as intended, $\exists p\, \mathit{UsedFor}(c, p)$ defines the set of coupons that are used.)

3. Any coupon that is used must be justified by sufficiently many purchased pizzas:

   $$\forall c[\mathit{Used}(c) \to \#(p, \mathit{Justifies}(c, p)) \geq \mathit{buy}(c)]$$

4. The number of pizzas any coupon is used for is not more than the number it allows us to get free:
$$\forall c[\#(p, \mathit{UsedFor}(c, p)) \le \mathit{free}(v)]$$

5. Each free pizza costs at most as much as the cheapest pizza used to justify use of the relevant coupon:
$$\forall c \forall p_1 \forall p_2[(\mathit{UsedFor}(c, p_1) \wedge \mathit{Justifies}(c, p_2)) \to \mathit{price}(p_1) \le \mathit{price}(p_2)]$$

6. We pay for every pizza used to justify use of a coupon:
$$\forall p \forall c[\mathit{Justifies}(c, p) \to \mathit{Paid}(p)]$$

7. The total cost is not too large:
$$\mathit{sum}(p, \mathit{Paid}(p), \mathit{price}(p)) \le k$$

8. We also must require that *Justifies*(c, p) and *UsedFor*(c, p) hold only of pairs consisting of a coupon and a pizza. (For example, if there are 3 coupons and 5 pizzas, we would not want *Justifies*$(4, 5)$ to hold, as we might then be getting pizza 5 free by applying the non-existent coupon number 4.)
$$\forall c \forall p[\mathit{Justifies}(c, p) \to (c \in [m] \wedge p \in [n])]$$
$$\forall c \forall p[\mathit{UsedFor}(c, p) \to (c \in [m] \wedge p \in [n])]$$
Because *Paid* and *Used* are defined in terms of *UsedFor*, it will follow that *Paid* contains only pizzas and *Used* contains only coupons.

4 Sequential Games

This problem involves reasoning about a sequence of steps, identifying a choice at each step so that the result of the sequence of steps satisfies a constraint. In the problem scenario, you are to play a sequence of games $G_1, G_2, \ldots, G_n$, under the following conditions:

1. You may play each game multiple times, but all plays of game G_i must be made after playing G_{i-1} and before playing G_{i+1}.
2. You pay 1 token each time you play a game, and you may play a game at most as many times as the number of tokens you have when you begin playing that game.

3. You must play each game at least once.
4. You have C tokens when you start playing G_1. After your last play of G_i, but before you begin playing G_{i+1}, you receive a “refill” of up to R tokens. However, you have some “pocket capacity” C, and you are never allowed to have more then C tokens.
5. Each game has a “fun” value for you, which may be negative.

Your goal is to decide how many times to play each game, to maximize your total fun. The problem can be described as follows.

Given: – Number $n \in \mathbb{N}$ of games;
– Fun value $v_i \in \mathbb{Z}$ for each game $i \in [n]$;
– Pocket Capacity $C \in \mathbb{N}$;
– Refill amount $R \in \mathbb{N}$;
– Goal $K \in \mathbb{N}$.

Find: A number of plays p_i for each game $i \in \mathbb{N}$ such that total fun is at least K.

Our instance vocabulary is (n, C, R, K, v), where v is a unary function and the other symbols are all constant symbols. A solution is a unary function p. In addition, to represent the constraints of the problem, we will use a unary funciton t, which gives the number of tokens in our pocket at start of playing each game. That is, just before we start playing game i, we have $t(i)$ tokens.

The constraints on t and p are:

1. $\sum_{i \in [n]} p_i v_i \ \geq \ K$:
$$sum(i, i \in [n], p(i) \cdot v(i)) \geq K$$
2. The number of tokens t_i available to play game i is C when we start playing the first game, and for $i > 1$ is the minimum of C and $t_{i-1} - p_{i-1} + R$:
$$t(1) = C \ \wedge \ \forall i[1 < i \leq n \rightarrow \exists x((x = t(i-1) - p(i-1) + R) \wedge \\ (x > C \rightarrow t(i) = C) \wedge (x \leq C \rightarrow t(i) = x))]$$
3. We play each game at least once, and at most t_i times:
$$\forall i[(1 < i \leq n) \rightarrow (1 \leq p(i) \leq t(i))]$$

5 Fixing an n-Queens Layout

This problem is interesting because it requires finding a sequence of steps that are more complex than in the sequential games problem. We begin with a collection of chess queens on an $n \times n$ chess board, some of which may be located so they are attacking each other. At each step we will move a queen, according to the usual rules, with the goal that after some number of moves no pair of queens will be attacking each other.

An instance structure will be for the vocabulary $[Pos, n]$, where

1. n is a constant symbol giving the length of one side of the board;
2. Pos is a binary relation symbol, with $Pos(x, y)$ meaning there is a queen at co-ordinates x, y.

Notice that we are not explicitly given the number of queens, and that we do not have identifiers for the queens. (In the pizza problem, we identified each pizza and each voucher with a natural number, so we could quantify over pizzas and vouchers by quantifying over the universe, and we could have functions mapping pizzas and vouchers to values. In this problem, we do not have domain elements for queens, so cannot quantify over them.)

We will write our specification entirely in terms of locations having queens (or not). Further, to describe the changes in the board as we move queens around, we will explicitly represent a sequence of board arrangements. That is, we will write formulas that define a class of structures in which a sequence of board arrangements is explicitly represented. For this purpose, we will use a ternary relation symbol P, with $P(i, x, y)$ meaning that the i^{th} board arrangement has a queen at co-ordinates x, y. We will introduce some other vocabulary symbols as we go, for convenience.

We will make the simplifying assumption that the universe is $[n]$, so the that quantified variables naturally range over board indices. A somewhat un-natural consequence of this is that we can have a sequence of at most n boards, and therefore are allowed only $n - 1$ queen moves, which might not be enough for some instances. (We could allow for more moves by using some arithmetic or by indexing boards by tuples of variables, at the cost of slightly more complex formulas.)

Our complete vocabulary is $[Pos, n, P, Attack, Move, LegalMove]$, where $P, Attack, Move, LegalMove$ are relation symbols of arities 3,4,5 and 5, respectively. The intuitive semantics is:

- $P(i, x, y)$ means there is a queen at x, y in the i^{th} board layout (or, at the i^{th} step of the plan).

- *Attack*(x_1, y_1, x_2, y_2) means that two queens at positions x_1, y_1 and x_2, y_2 attack each other.
- *Move*$(x_1, y_1, x_2, y_2, i$ meant that the transition from board layout i to board layout $i+1$ involves moving a queen from x_1, y_1 to x_2, y_2 (and no other changes).
- *LegalMove*(x_1, y_1, x_2, y_2, i) means that in board layout i, moving a queen from x_1, y_1 to x_2, y_2 is a legal queen move.

The requirements on the sequence of board arrangements are as follows.

1. The first board has the layout given by *Pos*:

$$\forall x \forall y [P(1, x, y) \leftrightarrow Pos(x, y)]$$

2. The last board must be a non-attacking arrangement:

$$\forall x_1 \forall y_1 \forall x_2 \forall y_2 [(((x_1 \neq x_2) \vee (y_1 \neq y_2)) \wedge (P(max, x_1, y_1) \wedge P(max, x_2, y_2))) \\ \rightarrow \neg Attack(x_1, y_1, x_2, y_2)]$$

   Now, we write a (somewhat long) formula defining when queens a two locations attack each other:

$$\forall x_1 \forall x_2 \forall y_1 \forall y_2 [Attack(x_1, y_1, x_2, y_2) \leftrightarrow (\\ ((x_1 = x_2 \vee y_1 = y_2) \vee \exists z [(x_1 = x_2 + z \wedge y_1 = y_2 + z) \ldots))]$$

   where the $\cdots$ must describe exactly the remaining pairs of diagonal attacking locations.

3. Exactly one queen moves at each step (that is, from a board layout i to board layout $i+1$).

   Since we have no identifiers for individual queens, we must state this in terms of locations. In particular, we will say that there is exactly one pair of locations with a queen moving from one location to the other, and for all other locations the presence (or not) of a queen does not change. We will write the formula using quantification over locations, which are pairs of co-ordinates. That is, we write formulas of the form $\forall \langle x, y \rangle \phi(\langle x, y \rangle)$.

$$\forall i [1 \leq i < n \ \rightarrow \exists! \langle x_1, y_1 \rangle \ \exists! \langle x_2, y_2 \rangle [Move(x_1, y_1, x_2, y_2, i) \wedge \\ \forall \langle x, y \rangle (((x \neq x_1 \vee y \neq y_1) \wedge (x \neq x_2 \vee y \neq y_2)) \rightarrow \\ (P(i, x, y) \leftrightarrow P(i+1, x, y)))]]$$

We now show that we can treat these non-standard quantifiers as abbreviations for formulas with standard quantifiers. We already use $\exists! x \phi(x)$ for $\exists x[\phi(x) \wedge \forall y \neq x \neg\phi(y)]$. The formula $\exists\langle x, y\rangle \phi$ naturally means $\exists x \exists y \phi$, so a formula of the form $\exists!\langle x, y\rangle \phi$ means

$$\exists x \exists y[\phi(x,y) \wedge \forall x_1 \forall y_1((x \neq x_1 \vee y \neq y_1) \rightarrow \neg\phi(x,y))]$$

We next define a queen moving from one location to another as the conjunction of appropriate changes in the existence of a queen at each location, plus what is required to be a legal queen move.

$$\begin{aligned}
&\forall 1 \leq i < n\ \forall x_1 \forall x_2 \forall y_1 \forall y_2 [\mathit{Move}(x_1, y_1, x_2, y_2, i) \leftrightarrow \\
&\qquad (P(x_1, y_1, i) \wedge \neg P(x_1, y_1, i+1) \wedge \neg P(x_2, y_2, i) \wedge P(x_2, y_2, i+1) \\
&\qquad\qquad \wedge \mathit{LegalMove}(x_1, y_1, x_2, y_2, i))]
\end{aligned}$$

We now need a formula giving the conditions under which it is legal to move a queen from one location to another:

$$\begin{aligned}
&\forall 1 \leq i < n \forall x_1 \forall x_2 \forall y_1 \forall y_2 [\mathit{LegalMove}(x_1, y_1, x_2, y_2, i) \leftrightarrow [\\
&\qquad (x_1 = x_2 \wedge y_1 < y_2 \wedge \forall y(y_1 < y \leq y_2 \rightarrow \neg P(i, x_1, y))) \vee \\
&\qquad (x_1 < x_2 \wedge y_1 = y_2 \wedge \forall x(x_1 < x \leq x_2 \rightarrow \neg P(i, x, y_1))) \vee \\
&\qquad \vdots \\
&\qquad\qquad\qquad]
\end{aligned}$$

6 Tetromino Tiling

...