Using Constraint Logic Programming to Analyze the Chronology in a William Faulkner Story

Jennifer Burg
Dept. of Mathematics and Computer Science
Wake Forest University
Winston-Salem, NC 27109
burg@mthcsc.wfu.edu

Sheau-Dong Lang
School of Computer Science
University of Central Florida
Orlando, Florida 32816
lang@cs.ucf.edu

Abstract

Constraint logic programming is a family of programming languages that allow for declarative problem statements including the expression of constraints in specialized domains. In CLP(*R*), for example, a problem is expressed in the logical format of a Prolog program, augmented with constraints that take the form of equations and inequalities over the real-number domain. In this paper, we apply CLP(*R*) to an analysis of the chronology of "A Rose for Emily" by William Faulkner, where the story is told in a non-chronological time sequence with incomplete information about relative or absolute points in time. Our analysis shows that Faulkner gives conflicting time information in the story. We come to this conclusion by expressing the time information as a system of constraints in CLP(*R*), sorting the time points in these constraints, and using a specialized constraint solver that identifies a "conflict set" when a conflict in the constraint system is uncovered.

Constraint Logic Programming

Constraint logic programming (CLP) is a family of programming languages based upon predicate logic augmented with constraint satisfaction in specialized domains. Like Prolog, the best-known language in the family, each CLP language allows the programmer to express a problem declaratively rather than procedurally. That is, a program is a statement of what must be true of all the variables in the problem for the solution to be attained. This problem statement is augmented with the expression of constraints in domains such as integers, rational numbers, or Boolean values. In CLP(*R*), for example, the problem can include equations and inequalities that express constraints among the program variables, which can take on any real-number values. In Chip, ECLiPSe, and CLP(*FD*), the variables take on values from discrete, finite domains, and the constraints might be expressed as ranges of integer values or llists of allowable value assignments for variables [Dincbas et al.; Van Hentenryck].

For each language in the family, a solution is found through a combination of resolution-based inference and constraint satisfaction. As execution proceeds, a constraint "solver" continually checks the satisfiability of the system of constraints it has encountered in the program so far. If the constraints are satisfiable, it is possible to assign values to all the variables such that all the constraints remain true statements. If the constraints are unsatisfiable, the program fails, since a solution is impossible.

CLP languages are valued for their natural, declarative expression of problems. They have been used to solve problems such as circuit design, stock portfolio management, warehouse location, resource allocation, and a variety of discrete optimization applications. In this paper, we use CLP(*R)* to analyze the chronology of a

William Faulkner short story. To our knowledge, this is the first demonstration of the language's applicability to an analysis of literature.

CLP(*R*) Applied to Time Elements in a Faulkner Short Story

As anyone who has attempted to discuss a Faulkner short story with a group of college freshmen can tell you, Faulkner chronologies can be challenging to new readers. The difficulty is that his stories are often not told in a straight beginning-to-end time sequence. "A Rose for Emily," for example, begins with Emily's death and meanders back and forth through a period of change from the pre-Civil War south to the modern age. It can be argued that there are thematic reasons for this winding. But for students who at first simply want to "get" the story, the initial task is to understand what happened, and when. It was this need to sort things out for our students that motivated us to consider the usefulness of CLP(*R*). (We should note that the first author of this paper taught college English for ten years before becoming a computer scientist.)

Part of the power of CLP(*R*) lies in its ability to verify that values are possible for all program variables in the context of existing constraints without requiring that the variables have *specific* values assigned to them. This makes it possible to apply CLP(*R*) to the sorting problem that is of interest to us. In an ordinary procedural programming language such as C or Pascal, sorting is done on specific values that have inherent ordering and can be judged as greater than, equal to, or less than each other in some sense -- numbers or letters of the alphabet, for example. But what if we wish to sort variables whose values are unknown at the time of the sort -- points of time about which we have only partial information regarding their relative positions? With CLP(*R*), we can express time-related information in the form of equations and inequalities where points in time are either constants or variables. For example, the fact that Emily's taxes were remitted in 1894 is expressed as $C = 1894$; and the fact that, with the exception of Emily's manservant, no one had entered her house for at least 10 years prior to her death comes out as $A - B \geq 10$. These points in time can then be sorted, perhaps with more than one ordering possible.

Another advantage of using CLP(*R*) for this purpose is the ability to uncover conflicts in constraint systems, and to pinpoint the cause of the conflict. It has been shown that when standard Gaussian elimination and the simplex method are used to check the satisfiability of the constraints, a minimal conflict set can be identified directly. This information can be used as the basis of intelligent backtracking in the solution of CLP(*R*) problems [Burg, Lang, and Hughes].

The first implementation of CLP(*R*) emerged from a research team at Monash University in Australia, and from there it evolved to a compiled version put out by IBM's Thomas Watson Research Center [Jaffar et al.]. Neither of these versions of CLP(*R*) had the conflict-identification feature described above. We have implemented our own constraint solver with conflict-identification and intelligent backtracking, and it is this CLP(*R*) implementation that we have applied to the "Rose for Emily" chronology problem.

Initially, our motivation for applying CLP(*R*) to the Faulkner story came from two directions: From the English professor's point of view, it provided a tool for sorting out the events of "A Rose for Emily" so that the story could be more easily explained to students. From the logic programmer's point of view, it gave us another interesting

application of CLP, where both sorting of non-ground values and identification of inconsistencies are possible. Our application of CLP(*R*) to "A Rose for Emily," however, gave us surprising results. Though we knew that inconsistencies could be uncovered by the intelligent backtracking solver, we did not expect to find one in Faulkner's non-chronological story-telling. But indeed a conflict was there.

Further research inspired by our experiments has revealed to us that we were not the first to wrestle with the chronology of "A Rose for Emily." A number of time sequences for the story have been offered in literary publications, the first in 1958. (See [Moore] for an overview of these.) Scholars have been debating their various chronologies for more than forty years now.

In what follows, we will describe the clues about time which Faulkner has given in this story, show how these pieces of information can be expressed as constraints, and show how the CLP(*R*) program for sorting the time elements identifies a conflict in the chronology of the story. We offer here a tool to facilitate the evaluation of consistent chronologies, or determine the source of conflict if they are inconsistent.

The Chronology of "A Rose For Emily"

If we were to tell the story of Emily Grierson, but with none of Faulkner's artistry, it would go something like this: *Emily Grierson was born in the early 1840's, the only daughter of a once-prominent family of the Old South. As a young woman, she appeared to have many suitors, but for some reason, Emily never married. Maybe she was too much under the control of her domineering father. Maybe the Griersons held themselves a little too high above the rest of the town, and none of the suitors were considered good enough for Emily. Maybe none of the suitors ever actually asked Emily to marry him. In any case, Emily's father died and left her with almost nothing, and when Emily reached 30 without marrying, the townspeople actually began to pity her a little. Then Homer Barron, a construction worker, came to town. Before long, Emily was seen everywhere with Homer. He wasn't the kind of man the townspeople would have expected Emily to marry -- a common laborer and a Yankee -- and the propriety of her unchaperoned relationship with him was even questionable. Emily's out-of-town cousins were called in to save Emily's reputation. While they were in town, Emily went to the druggist and bought some arsenic, and everyone feared that she may try to kill herself. But then she bought a man's toilet set with the initials H.B. on each piece, and it looked like a wedding was in the offing. The wedding never materialized. Homer disappeared and was never seen again. Emily became reclusive after that. She was sometimes seen through the window, like the time when some men were sent out to sprinkle lime around her house because it smelled so bad. For about eight years, Emily gave China painting lessons to earn some money. Once, a deputation of town aldermen was sent to her house to tell her that she had to pay taxes. But other than that, no one but her manservant entered her house. On the day of her funeral, the townspeople finally got to see the inside of her house again. This is when they found out what happened to Homer Barron. When they entered Emily's bedroom, there lay Homer's skeleton on the bed.*

Faulkner does not tell this story in chronological order. However, if one pays close attention to information about time, all the clues are there. At the time of Emily's death, no one "had seen [the inside of her house] for at least ten years." "In 1894...Colonel Sartoris...remitted her taxes...." A deputation of aldermen from "the next

generation" then called on her to tell her the deal was off. At that time "Colonel Sartoris had been dead almost ten years." "She had vanquished their fathers thirty years before about the smell." And so forth.

Table 1 below gives the variables to be used in the time constraints along with their meanings. Table 2 gives the constraints, with the corresponding information from which each constraint was inferred. Figure 1 gives the CLP(R) program, consisting of the constraints followed by a *sort* predicate.

The Constraint Satisfaction Algorithm and Conflict Sets

In what follows, we assume the reader has a basic knowledge of Prolog-like languages, and refer the reader to [Clocksin and Mellish] and [Sterling and Shapiro] for details.

The execution of a CLP(R) program can be divided into two components: resolution-based inferencing coupled with unification for the binding of variables, in the manner first applied to Prolog [Kowalski; Robinson]; and constraint satisfaction in the real-number domain [Colmerauer; Cohen]. Thus, a CLP(R) interpreter can be divided into two corresponding components: an inference engine to perform a depth-first search through the program space; and a constraint solver to check the satisfiability of the collected constraints at the entrance to each predicate.

The basic implementation of a constraint solver for CLP(R) is given in [Jaffar et al.], to which we have added mechanisms for conflict identification and intelligent backtracking [Burg, Lang, and Hughes]. The task of the solver is to determine if the constraints collected thus far during execution are satisfiable. That is, given the bindings performed during unification, is it possible to assign real-number values to the remaining program variables such that all constraints simultaneously hold true? (We should note that in both the *Jaffar et al.* implementation and our own, the constraints are limited to linear equations and inequalities for efficiency reasons). The constraints arise from one of two sources -- either from arithmetic expressions (containing variables) that are equated during unification, or from equations or inequalities in the bodies of program clauses. Since more constraints are added to the system each time a clause is entered during program execution, constraint satisfaction is necessarily incremental.

More formally, the solver's problem is as follows: Say that for each inequality constraint

$$a_1x_1 + \ldots + a_nx_n \le b \text{ (or, } a_1x_1 + \ldots + a_nx_n \ge b)$$

the inequality is converted to an equation of the form

$$a_1x_1 + \ldots + a_nx_n + s = b \text{ (or, } a_1x_1 + \ldots + a_nx_n - s = b)$$

where the slack variable s is assumed to have a non-negative value. (This can be easily generalized to strict inequalities as well, but we will not do so here for simplicity of notation.) At any moment in execution, we can assume we have a satisfiable system of the form

$$M \cdot X = B \qquad (1)$$

where $M = \begin{bmatrix} 1 & * & \ldots & \ldots & * \\ 0 & 1 & \ldots & \ldots & .. \\ \ldots & \ldots & 1 & \ldots & \ldots \\ 0 & \ldots & 0 & 1 & * \end{bmatrix}$ is an $m \times n$ matrix ($m \le n$)

for m equations and n unknowns, $x = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$ represents the n unknowns, and $B = \begin{bmatrix} b_1 \\ \dots \\ b_m \end{bmatrix}$ represent the constants on the righthand side of the equations, with all $b_i \geq 0$.

Notice that the unknowns $x_1, \dots x_m$, corresponding to the 1's on the diagonal of the matrix M in (1), are the basic variables. The system (1) is satisfiable because an obvious solution is obtained by letting the basic variables $x_i = b_i$ for $1 \leq i \leq m$, and letting the non-basic variables $x_{m+1} = \dots = x_n = 0$.

Given such a system, then time a new constraint is encountered during program execution, the solver's problem is to check the satisfiability of the new system, including the new constraint.

In [Burg, Lang, and Hughes], we describe an incremental version of Gaussian elimination combined with the simplex method for checking the satisfiability of the system of constraints. All operations in this algorithm are basic row operations where a multiple of one row (i.e., constraint) is added to another as we determine which variable to "solve for" in each constraint. In effect, we keep substituting variables out of the last row of our constraint system until we arrive at one of two results: If the system (including the new constraint) is indeed satisfiable, then either the new constraint is completely eliminated because it is redundant, or the system is transformed into the same form as in (1), with one extra row added. If, on the other hand, the system is unsatisfiable when the new constraint is added to it, then our row operations has put the last row in the form

$$a_{m+1,m+1} x_{m+1} + \dots + a_{m+1,n} x_n = b_{m+1}$$

where $a_{m+1,i} \leq 0$ and $b_{m+1} > 0$. This last equation clearly cannot be satisfied by the unknown $x_i \geq 0$ for $1 \leq i \leq n$, because each coefficient $a_{m+1,i} \leq 0$ but the constant $b_{m+1} > 0$.

To identify a set of conflicting constraints in the system, we need to keep a record of the row operations performed during Gaussian elimination and the simplex procedure. Specifically, if we let matrix R represent the set of equations in their original form, each time a row operation is applied to an equation in R, the matrix R is transformed into $E \cdot R$, where E is a matrix corresponding to the row operation. Therefore, we could use a matrix B which is equal to the product of these E matrices to record the successive row operations. As a result, the current coefficient matrix M in the transformed system (as in (1)) is related to the matrix R by the simple equation

$$M = B \cdot R$$

It has been proved in [Burg, Lang, and Hughes] that the indices of the non-zero entries in the last row of matrix B identify exactly the rows (i.e., constraints) in R which cause the conflict. We have shown that the conflict set revealed by our intelligent backtracking solver is in fact a minimal conflict set -- that is, if we remove any one constraint from the

set, it is no longer inconsistent. (However, we should note that the conflict set revealed by the algorithm is not necessarily unique.)

The Conflict in "A Rose for Emily"

If you attempt to run the program given in Figure 1, you'll find that it never actually gets to the *sort* procedure. This is because the constraint solver discovers a conflict when it reaches the constraint $E = B$. Our own CLP(*R*) interpreter lists the conflicting constraints as follows:

$$B \geq D$$

$$D - E \geq 8$$

$$E = B$$

Intuitively, it should be clear that these constraint are irreconcilable. Faulkner tells us that the deputation of aldermen went *into* Emily's house:

> A deputation waited upon her, knocked at the door through which no visitor had passed since she ceased giving china-painting lessons eight or ten years earlier. They were admitted by the old Negro into a dim hall from which a stairway mounted into still more shadow.

This we represent as $D - E \geq 8$ (and $D - E \leq 10$). Common sense then dictates that the last time anyone saw the inside of Emily's house (moment B) was *at the same time or after* the visit of the aldermen, i.e., $B \geq D$. In the same passage we learn that the china-painting lessons occurred between eight and ten years earlier. Later in the story we are told that the china-painting students were the last townspeople to enter Emily's house:

> Then the newer generation became the backbone and the spirit of the town, and the painting pupils grew up and fell away and did not send their children to her with boxes of color and tedious brushes and pictures cut from the ladies' magazines. The front door closed upon the last one and remained closed for good.

This we translate into the constraint $E = B$.

One way to reconcile this conflict is to assume that when Faulkner says that "the front door closed … for good," he means that only in the context of the China painting lessons. In any case, *some* re-interpretation is necessary in order to arrive at a consistent chronology.

Removing the Conflict and Sorting the Events of the Story

Since the conflict set uncovered by our CLP(*R*) interpreter is a *minimal* conflict set, we can resolve it by removing any one of the constraints. In this case, we can remove the constraint E = B, and it is then possible to sort the events of the story.

The complete CLP(*R*) program is given in Figure 1. With the constraint $E = B$ left in, the solver never executes the sort because it finds a conflict in the constraint set. With the constraint $E = B$ deleted, the solver finds five solutions (disregarding duplicate solutions that swap the position of time points that can be equal). The differences among them arise from the overlapping spans of time within which events might fall. The time line below gives a representative ordering of the variables, with feasible dates filled in.

This sorting exercise and an examination of the resulting timeline can be meaningful to students trying to understand Faulkner's work, for through the sorted

timeline we see more clearly the transitional time period during which the events of the story take place. In our sample timeline, Emily was born in 1850 and died in 1924, her life beginning before the Civil War and ending within America's period of industrialization and growth. Our interpretation of events places Emily's birth earlier than many others would place it. (In fact, our constraints accommodate a birthdate as early as 1842.)

Faulkner's Treatment of Time

One might like to have some explanation for the apparent slip in Faulkner's story-telling. But the question of whether or not Faulkner simply made a mistake is not the central issue in our understanding of Faulkner's treatment of time. It is the narrator, not Faulkner, who is telling us Emily's story. And like the old men in their brushed Confederate uniforms lined up for Emily's funeral, perhaps he is "confusing time with its mathematical progression, as the old do, to whom all the past is not a diminishing road but, instead, a huge meadow which no winter ever quite touches…." If we pick events from this meadow of recollections and then try to put them in a sequence, perhaps it isn't surprising after all that they don't line up perfectly.

Useful Extensions and Applications of this System

We have shown that CLP(*R*) is a system useful for the chronological sorting of events in stories where events are related out-of-order. It is easy to play with numbers, adding equality constraints to the program to determine if certain dates for certain events are possible. With this system, it is possible to compare different interpretations of the story's chronology, find inconsistencies, and see more clearly the range of dates within events might fall.

However, this application of CLP(*R*) does require that the reader transform events and their relative order into the kinds of constraints shown in Tables 1 and 2 – perhaps not a bad thing, since it values the human input to the interpretive process. To automate this part of the process would be a challenging problem of artificial intelligence. (Imagine, for example, an intelligent program that could interpret even a straightforward sentence like this one: "The whole town went to her funeral…to see the inside of her house, which no one save an old manservant…had seen in at least ten years.")

Extensions of this system that are feasible, however, would include providing a user interface through which the reader could, graphically, (1) define points in time, (2) indicate which points are before which others, (3) give dates where dates are known, (4) ask for a sort, 5) see a graphical presentation when the sort is possible, or (6) receive a list of conflicting constraints when a sort is not possible. A reasonable level of automated intelligence might also be realized by building in certain events with related common sense knowledge. Common sense constraints are so "taken for granted" by readers that they might often be omitted in the human's enumeration of time constraints, resulting in nonsensical sorts. For example, Homer Barron's disappearance must have happened *after* he came to town (J > I). Even more obviously, a person must be born before he or she dies – in fact, all events of a person's life must precede death. Built in "birth" and "death" time points could be automatically accompanied with these common sense constraints.

Our current research involves the building of such a user interface, along with testing the system on more complex works of fiction. We have shown the usefulness of constraint logic programming on short works, to help in explaining non-chronological story-telling and to compare different interpretations of the chronology. On longer, more complex works, applying constraint programming techniques would be a more time-consuming project, but one with great potential value for pedagogy and literary research.

Variable	Meaning
A	Emily's death
B	Last time anyone but Emily's manservant saw the inside of her house
C	Colonel Sartoris remitted Emily's taxes
D	A deputation called on Emily asking her to start paying taxes again
E	Emily stopped giving China painting lessons
F	Colonel Sartoris died
G	There was a bad odor around Emily's house
H	Emily's father died
I	Homer Barron came to town
J	Homer disappeared
K	Emily was born
L	Emily appeared in town again after Homer's disappearance
M to N	First period of time when Emily shut her doors to the public
N to E	Period of China painting lessons
E to A	Second period of seclusion

Table 1 Variables and Their Meaning

Constraint	Meaning
A-B>=10	No one besides the manservant had seen the inside of Emily's house for at least 10 years before her death.
C=1894	Colonel Sartoris remitted Emily's taxes in 1894.
D–C>=10 D–C<=20	A generation later, a deputation called on Emily to tell her that she would have to pay taxes after all.
D–E>=8 D–E<=10	Eight or ten years passed between the time when Emily last gave China painting lessons and the time the deputation called on her.
F>C	Colonel Sartoris died after he remitted Emily's taxes.
D–F>9 D–F<10	Colonel Sartoris died almost 10 years before the deputation called on Emily.
D–G=30	There was a bad odor around Emily's house 30 years before the deputation called on her about her taxes.
B>=D,	The last time anyone but the manservant saw the inside of Emily's house had to be after or at the same time as the visit of the deputation
B>=E	The last time anyone but the manservant saw the inside of Emily's house had to be after or at the same time as the last China painting lesson
G–H=2	The odor around Emily's house appeared two years after her father's death.
I>H	Homer Barron came to town after Emily's father died.
I–K > 30	Emily was older than 30 when Homer Barron came to town
J>I	Homer disappeared after he came to town.
G–J<0.5 G>J	The odor appeared less than 6 months after Homer disappeared.
L–J < 0.5	Emily appeared again on the streets after Homer's disappearance.
A–K=74	Emily died at the age of 74.
M>L	The first period when Emily shut her doors to the public happened after her reappearance after Homer's death.
E–N>=6 E–N<=7	Emily gave China painting lessons for 6 or 7 years.
E=B	When Emily shut the door on her last China painting student, no one but her manservant saw the inside of her house after that.
N–K<=45 N–K>=38	Emily was about 40 when she gave China painting lessons.
C>=N C<=E	Colonnel Sartoris remitted Emily's taxes while the China painting lessons were going on.
M<N	The beginning of the period of seclusion has to be before the end.
A>E, A>K, Etc.	Emily died after everything else.

Table 2 Constraints and Their Meaning

```
emily([A,B,C,D,E,F,G,H,I,J,K,L,M,N], LIST):-
      A-B >= 10,
      C = 1894,
      D-C>=10, D-C<=20,
      D-E>=8, D-E<=10,
      F>C,
      D-F>9, D-F<10,
      D-G=30,
      B >= D, B >= E,
      G-H=2,
      I>H,
      I-K>30,
      J>I,
      G-J<0.5,
      G>J,
      L-J=0.5,
      A-K=74,
      M>L,
      E-N>=6, E-N<=7,
      %E = B,    /*This constraint inserts a conflict*/
      N-K<=45,
      N-K>=38,
      C>=N, C<=E,
      M<N,
      A>E, A>K,
      insort([A,B,C,D,E,F,G,H,I,J,K,L,M,N], LIST).

insort([],[]).
insort([X|L],M):-
      insort(L,N), insortx(X,N,M).
insortx(X, [A|L], [A|M]):-
      A<=X, insortx(X, L, M).
insortx(X,L,[X|L]):-
      X<=A, starts(A,L).
insortx(X,[],[X]).
starts(A, [A|L]).
```

Figure 1. The CLP(R) Program

K (1850)	Emily is born
H (1879)	Emily's father dies
I	Homer Barron comes to town
J	Homer disappears
G (1881)	A bad odor appears around Emily's house
L	Emily reappears after a period of seclusion
M	Emily begins a second period of seclusion
N (1894)	Emily ends second period of seclusion; begins giving China painting lessons
C (1894)	Emily's taxes are remitted
E (1901)	Colonel Sartoris dies
F	Emily stops giving China painting lessons
D (1911)	A deputation of town officials call on Emily about her taxes
B (1914)	Last time anyone but Emily's servant sees the inside of her house
A (1924)	Emily dies at the age of 74

Figure 2. The Timeline for "A Rose for Emily"

References

Burg, J., S.-D. Lang, and C. E. Hughes. Intelligent Backtracking in CLP(*R*). *Annals of Mathematics and Artificial Intelligence* 17 (1996), 189-211.

Clocksin, W.F., and C. S. Mellish. *Programming in Prolog.* 3rd ed. New York: Springer-Verlag, 1987.

Cohen, J. Constraint Logic Programming Languages. *Communications of the ACM* 33, 7 (1990), 52-68.

Colmerauer, A. An Introduction to Prolog III. *Communications of the ACM 33, 7* (1990), 69-90.

Dincbas et al. The Constraint Programming Language CHIP. *International Conference on First Generation Computing Systems*, Tokyo, Japan, November 1988.

Faulkner, W. *Collected Stories of William Faulkner*. New York: Random, 1950.

Going, William T. Chronology in Teaching "A Rose for Emily." Reprinted in Inge, 76-83.

Inge, M. Thomas. *William Faulkner: A Rose for Emily*. The Charles Merrill Literary Casebook Series. Columbus, Ohio: Merrill, 1970.

Jaffar, et al. The CLP(*R*) Language and System. *ACM Transations on Programming Languages and Systems* 14, 3 (July 1992), 339-395.

Kowalski, R. Algorithm = Logic + Control. *Communications of the ACM* 22, 7 (1979), 424-436.

McGlynn, P. D. The Chronology of "A Rose for Emily." Reprinted in Inge, 90-92.

Moore, G. M. Of Time and its Mathematical Progression: Problems of Chronology in Faulkner's "A Rose for Emily." *Studies in Short Fiction* 29 (1992), 195-204.

Nebeker, H. E. Chronology Revised. *Studies in Short Fiction* 8 (1971), 471-473.

Perry, M. Literary Dynamics: How the Order of a Text Creates its Meanings [With Analysis of Faulkner's "A Rose for Emily"]. *Poetics Today* 1, 1-2 (Autumn 1979), 35-64, 311-361.

Robinson, J. A. *Logic and Logic Programming. Communications of the ACM* 35, 3(1992), 40-64.

Sterling, L. and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. 2nd ed. Cambridge: MIT Press, 1994.

Van Hentenryck, P. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

Wilson, G. R., Jr. The Chronology of Faulkner's "A Rose for Emily" Again. *Notes on Mississippi Writers* 5 (Fall 1972), 56, 44, 58-62.

Woodward, R. H. The Chronology of "A Rose for Emily." Reprinted in Inge, 84-86.