The primary purpose of the final project is to provide you an opportunity to demonstrate your understanding of CSP and your ability to use machine-readable CSP and FDR to model and reason about nontrivial systems and situations.

You may work either alone or with one other person in the same version of the course as you (e.g., CIS 400). However, every person must write their own project report. Each of the suggested projects has a number of stars next to it, indicating how challenging I think it is. In some cases, the challenge lies not in the CSP itself but rather in understanding the scenario.

- If you are taking CIS 400 and will work alone, you may choose any project at all; if you will work with another CIS 400 student, you must choose a project with at least three stars.

- If you are taking CIS 632 and will work alone, you may choose any project with at least two stars; if you will work with another CIS 632 student, you must choose a project with four stars.

What I expect you to turn in

- An annotated machine-readable CSP script that includes all necessary definitions and assertions for your project. Thus, it should contain the system itself, the properties it is supposed to satisfy, and the assertions (e.g., refinement or deadlock checks) necessary for validating those properties.

  You will need to submit both a paper copy and an electronic copy of this script. Annotations should include high-level discussion of the CSP code, as well as any additional information necessary to understand how things fit together. It is your responsibility to make sure that I can understand how your code works.

- A typewritten (computer-generated) paper—roughly, in the 8–10 pages range—that includes a high-level description of both the problem you’re trying to solve/analyze and your CSP solution. You should also include some type of analysis of your results: describe your use of refinements (e.g., explain why you use trace refinements instead of failures refinements or vice versa), any unexpected results, anything you would do differently as a result of your experience, etc. If your CSP code does not work completely, be sure to describe its shortcomings and where the problems lie. Remember that I am looking for evidence that you understand what’s going on: a clear and insightful discussion of your code’s shortcomings (if they exist) will provide that evidence.

  This paper will be graded for grammar, spelling and style, as well as for content. As always, provide proper citations when building on other’s work or ideas.

The project is officially due on the last day of class (December 5). However, I will accept projects through Monday, December 16.

Suggested Projects

I am very open to other suggestions for possible projects (I had trouble coming up with many). If there is something else you’d like to model, feel free to discuss it with me. However, do not work on an unapproved project, as you may be unhappy with the results. Some of the suggestions are fairly vague: come see me to get more details of what I have in mind, as many are based on unpublished practicals given at another university.

Fall 2002
CSP as a Puzzle Solver

1. (⋆) There is a train-shunting puzzle in David Wells’ *The BPenguin Book of Curious and Interesting Puzzles*, which involves a matter of moving trains into particular sheds. Use CSP and FDR to solve this puzzle. (See me for more details.)

2. (⋆⋆) Section 15.1 of Roscoe’s text [Ros98] describes the Peg Solitaire puzzle we discussed in class. Write a puzzle solver for the extended version described in Exercise 15.1.3 of that text, and solve at least the two puzzles mentioned there. Also do Exercise 15.1.2 of Roscoe’s text. Set it up generally enough that the numbers of the particular items present can be easily changed. You may have seen this puzzle before and already know the solution: do not simply write CSP that encodes that solution.

3. (⋆ ⋆) Another sort of common puzzle involves a board with lights and switches, where pressing a switch has some effect on certain lights (i.e., turning them on or off). Use CSP to set up a framework for describing and solving these types of puzzles. (Again, come see me for more details.)

4. (⋆⋆) The game of chess is played on an $n \times n$ grid (in actually, $n = 8$, but let’s keep it more general here). In chess, knights move in a peculiar fashion: they move two spaces in one direction (horizontally or vertically) and one space in an orthogonal direction (vertically or horizontally). Thus, a knight at position $(i, j)$ can move to one of eight possible positions (assuming none involves him leaving the board): $(i + 2, j + 1), (i - 2, j + 1), (i + 2, j - 1), (i - 2, j - 1), (i + 1, j + 2), (i + 1, j - 2), (i - 1, j + 2), (i - 1, j - 2)$.

A knight’s tour is a sequence of valid knight moves in which the knight begins at some square on the grid and travels to each square exactly once, ultimately arriving back at the initial square. The web site http://www.BordersChess.org/KnightTour.htm gives an example of such a tour.

Use CSP to find knight’s tours (or determine that they don’t exist) for a variety of choices of $n$. Note that the runtime gets large as $n$ increases, so start out small: you may also find it useful to use some tactics for larger $n$. Include in your analysis some description of the runtimes necessary to check out the different board sizes (i.e., at what point does your solution start becoming infeasible?).

5. (⋆ – ⋆ ⋆ ⋆) Other puzzles?

Security-related protocols

1. (⋆⋆⋆) Read Lowe and Roscoe’s paper on detecting errors in the TMN protocol [LR97]. Implement the initiator, responder, and intruder described in Section 5 for the second version of the protocol. You should be able to find attacks 5.1, 5.2 and 5.3 using your implementation. Be careful: for the sake of efficiency, you will need to define the intruder differently than it appears in the paper.

2. (⋆⋆⋆) Implement the initiator, responder and intruder for the corrected version of the Needham-Schroeder Protocol, as given in Lowe’s paper [Low96]. You should be able to verify that the system indeed meets the specification correctly. Be careful: for the sake of efficiency, you will need to define the intruder differently than it appears in the paper.

Warning: Although some students have tried this in the past, I’ve yet to have someone reduce the state space enough to make running FDR on it feasible. You should be prepared to spend some time being clever with the code given in the paper. I have some suggestions, but I haven’t tried them.
3. (⋆ – ⋆ ⋆ ⋆⋆) Other security protocols?

**Assorted Algorithms and Protocols**

1. (⋆ ⋆ ⋆) In the context of distributed systems, an election is a procedure to select one process out of a group of processes. Elections typically arise when one member of a process group fails—the winner of the election takes over the responsibilities of the failed process.

   The Bully algorithm is an election algorithm that can be used when all processes know each other’s identities and addresses. Model the Bully algorithm in CSP. (See me for some hints on how to go about this.)

2. (⋆ ⋆ ⋆) Another possibility for elections is a ring-based election algorithm, which is suitable when processes are arranged in a ring and each process knows only its successor’s address/identity. Model the ring-based algorithm described by Coulouris and co-authors [CDK94].

3. (⋆ ⋆ ⋆) In distributed transactions, it is often necessary to gain consensus across several processors. For example, if Alice deposits a check from Bob, her bank shouldn’t credit the funds to her account without Bob’s bank debiting his account (or vice versa). The two-phase protocol, also described in [CDK94], provides a mechanism for a transaction to be aborted if any participant in the transaction aborts.

   Model the two-phase protocol in CSP. Since you need to account for possible processor failures/timeouts here, the above mentioned hints for the Bully algorithm may also be helpful; see me if you’re interested.

4. (⋆ ⋆ ⋆) Schneider’s textbook [Sch00] discusses a distributed-sum algorithm in Sections 5.4. Implement and verify the distributed-sum network, validating the given correctness properties. You should set it up to be fairly general: just as the Peg Solitaire solver could handle any rectangular board, your system should be set up to handle any finite graph. Note: You will probably want to initially try your network out with a very small graph. Once it’s working, you can try out larger graphs. As part of your analysis, include mention of what sizes of graphs are feasible and which ones are too big for “reasonable time” solutions.

5. (⋆ ⋆ ⋆) The distributed-sum algorithm is discussed further in Section 7.5 of Schneider’s text, where he discusses liveness properties.

   Implement and verify the distributed-sum network, validating the given correctness properties as above as well as the liveness properties of Section 7.5.

6. (⋆ – ⋆ ⋆ ⋆⋆) Other protocols, including communication protocols, distributed protocols, etc?

**Other Systems**

1. (⋆ ⋆) Specify and implement a model of an elevator, incorporating a notion of timing into the model. (See me for details.)

   Note: This one is not so much hard as it is rather long.

2. (⋆ – ⋆ ⋆ ⋆⋆) Other systems?
References


