ECE570 Lecture 10: Constraints

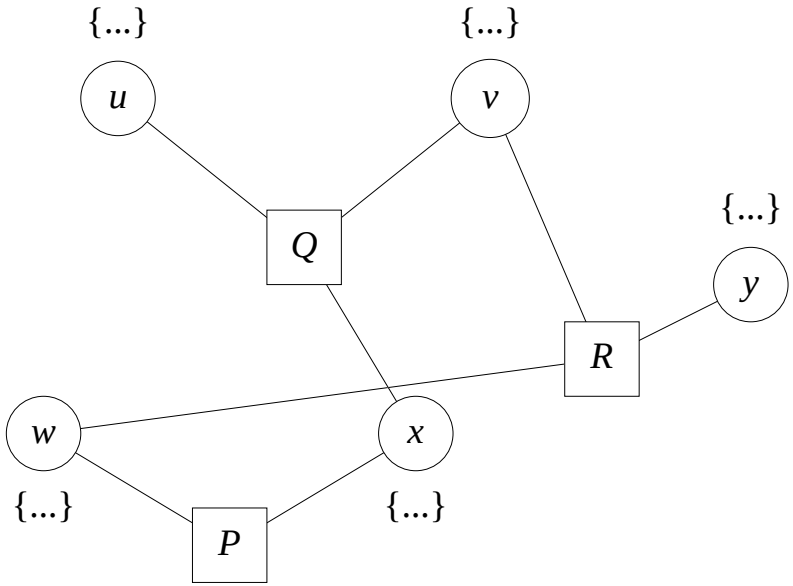
Jeffrey Mark Siskind

School of Electrical and Computer Engineering

Fall 2013

PURDUE
UNIVERSITY

CSPs as Directed Hypergraphs

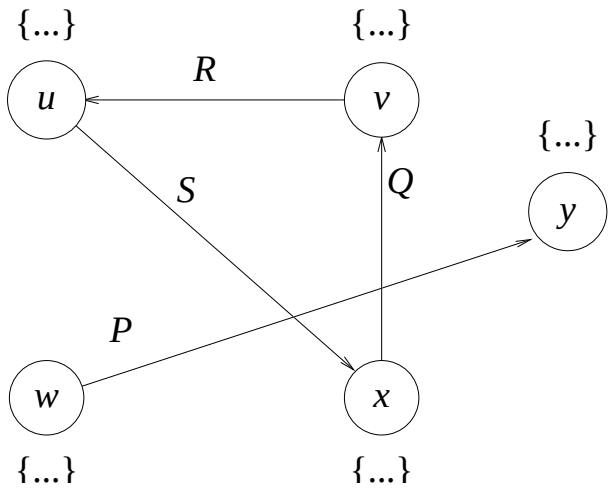


variables correspond to vertices

vertices labeled with domains

constraints are directed hyperedges

Binary CSPs as Directed Graphs



variables correspond to vertices

vertices labeled with domains

constraints are directed edges from first argument to second argument

Encode CSP as SAT

Variables and their domains:

$$x \in \{a_1, \ldots, a_n\}$$

are encoded as a disjunction:

$$x = a_1 \vee \cdots \vee x = a_n$$

A constraint such as:

$$P(x, y, z)$$

that is false for $x = a$, $y = b$, and $z = c$
is encoded as:

$$\overline{x = a \wedge y = b \wedge z = c}$$

or equivalently as:

$$x \neq a \vee y \neq b \vee z \neq c$$

General Algorithm for Solving CSPs

1. Terminate if all variables are bound.
2. Choose an unbound variable x_i.
3. Choose a value $d \in D_i$ for x_i.
4. Bind $x_i \mapsto d$.
5. Perform inference.
6. Go to 1.

Issues

Terminate: only if producing a single solution.
Fail if wish to produce multiple solutions.

Choose: No need to backtrack over choice of unbound variable.

Choose: Need to backtrack over choice of value for variable.

Bind: must be done in such a way so that it is undone upon backtracking.

Perform inference: will vary depending upon technique.

Some Additions to Scheme

(set! x e)
(set-car! l e)
(set-cdr! l e)
(string-set! s i c)
(vector-set! v i e)

(local-set! x e)
(local-set-car! l e)
(local-set-cdr! l e)
(local-string-set! s i c)
(local-vector-set! v i e)

(define-structure *type* $s_1 \ldots s_n$)

(make-*type* $e_1 \ldots e_n$)
(*type*? *object*)
(*type*-s_i *instance*)
(set-*type*-s_i! *instance* e)
(local-set-*type*-s_i! *instance* e)

Solving CSPs in Scheme—I

```
(define-structure domain-variable
 domain
 before-demons
 after-demons)

(define (create-domain-variable domain)
 (when (null? domain) (fail))
 (make-domain-variable domain '() '()))
```

Solving CSPs in Scheme—II

```
(define (attach-before-demon! demon x)
 (local-set-domain-variable-before-demons!
  x (cons demon (domain-variable-before-demons x)))
 (demon))

(define (attach-after-demon! demon x)
 (local-set-domain-variable-after-demons!
  x (cons demon (domain-variable-after-demons x)))
 (demon))
```

Solving CSPs in Scheme—III

```
(define (restrict-domain! x domain)
 (when (null? domain) (fail))
 (when (< (length domain)
          (length (domain-variable-domain x)))
  (for-each (lambda (demon) (demon))
            (domain-variable-before-demons x))
  (local-set-domain-variable-domain! x domain)
  (for-each (lambda (demon) (demon))
            (domain-variable-after-demons x))))
```

Solving CSPs in Scheme—IV

```
(define (bound? x)
 (null? (rest (domain-variable-domain x))))

(define (binding x)
 (first (domain-variable-domain x)))

(define (a-member-of l)
 (when (null? l) (fail))
 (either (first l) (a-member-of (rest l))))
```

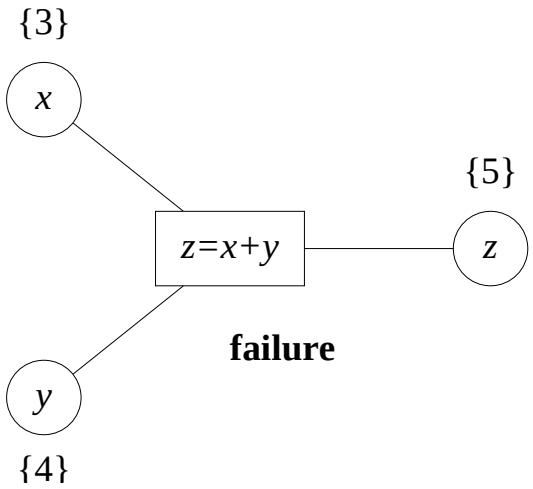
Solving CSPs in Scheme—V

```
(define (csp-solution domain-variables select)
 (define (loop domain-variables)
  (let ((domain-variables
         (remove-if bound? domain-variables)))
   (unless (null? domain-variables)
    (let* ((x (select domain-variables))
           (value (a-member-of
                   (domain-variable-domain x))))
     (restrict-domain! x (list value))
     (loop (removeq x domain-variables))))))
 (loop domain-variables)
 (map binding domain-variables))
```

Inference Techniques

- Early Failure Detection
- Forward Checking
- Value Propagation
- Generalized Forward Checking
- Node Consistency
- Arc Consistency
- Path Consistency

Early Failure Detection



If there exists some constraint such that

1. all of its arguments are bound and
2. the constraint is violated

then fail.

Early Failure Detection in Scheme—I

```
(define (assert-unary-constraint-efd! constraint x)
 (attach-after-demon!
  (lambda ()
   (when (bound? x)
    (unless (constraint (binding x)) (fail))))
  x))
```

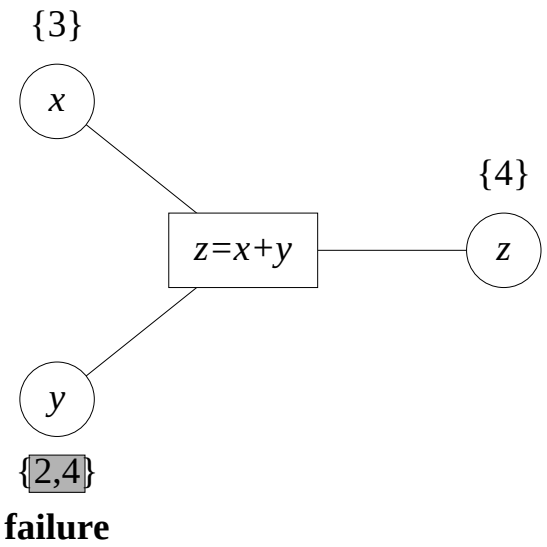
Early Failure Detection in Scheme—II

```
(define (assert-binary-constraint-efd! constraint x y)
 (for-each
  (lambda (v)
   (attach-after-demon!
    (lambda ()
     (when (and (bound? x) (bound? y))
      (unless (constraint (binding x) (binding y))
       (fail))))
    v))
  (list x y)))
```

Early Failure Detection in Scheme—III

```
(define (assert-ternary-constraint-efd! constraint x y z)
 (for-each
  (lambda (v)
   (attach-after-demon!
    (lambda ()
     (when (and (bound? x) (bound? y) (bound? z))
      (unless (constraint
                (binding x) (binding y) (binding z))
       (fail))))
    v))
  (list x y z)))
```

Forward Checking



If there exists some constraint such that

1. all but one of its arguments are bound and
2. the constraint is violated for all values of the unbound argument

then fail.

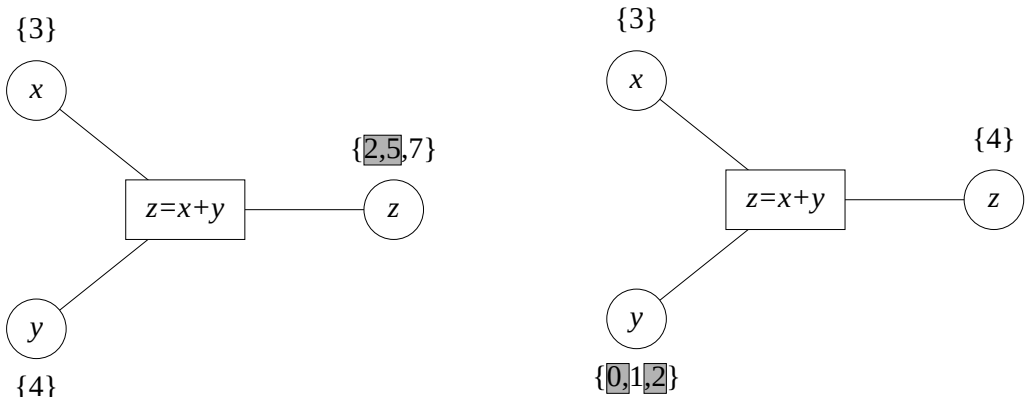
Forward Checking in Scheme—I

```
(define (assert-unary-constraint-fc! constraint x)
 (attach-after-demon!
  (lambda ()
   (unless (some (lambda (xe) (constraint xe))
                 (domain-variable-domain x))
    (fail)))
  x))
```

Forward Checking in Scheme—II

```
(define (assert-binary-constraint-fc! constraint x y)
 (for-each
  (lambda (v)
   (attach-after-demon!
    (lambda ()
     (when (bound? x)
      (unless (some (lambda (ye) (constraint (binding x) ye))
                    (domain-variable-domain y))
       (fail)))
     (when (bound? y)
      (unless (some (lambda (xe) (constraint xe (binding y)))
                    (domain-variable-domain x))
       (fail))))
    v))
  (list x y)))
```

Value Propagation



multidirectional

If there exists some constraint such that

1. all but one of its arguments are bound and
2. the constraint is violated for all but one of the values of the unbound argument

then bind the unbound argument to that value.

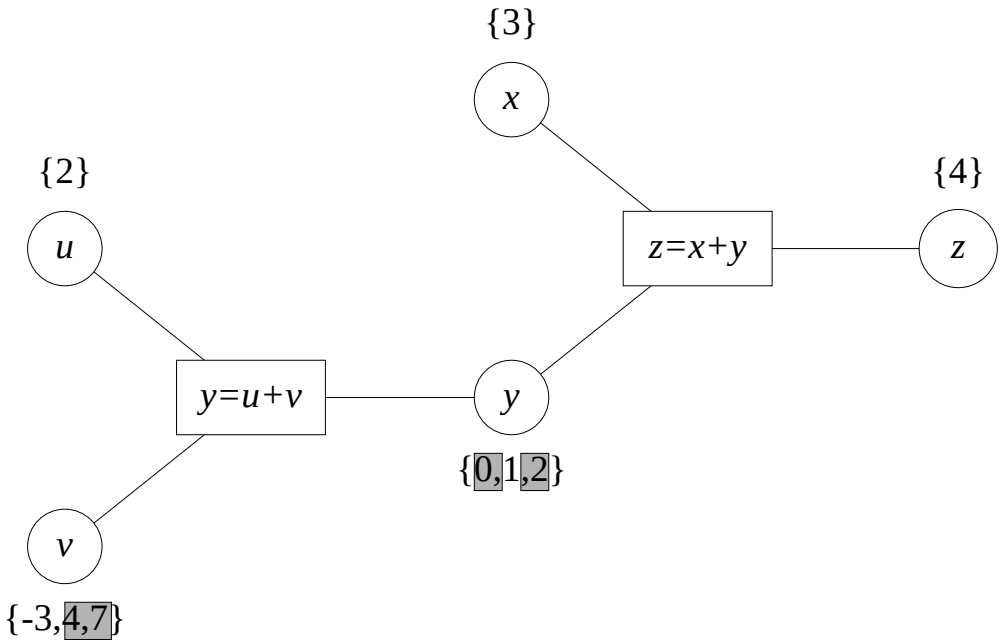
Value Propagation in Scheme—I

```
(define (assert-unary-constraint-vp! constraint x)
 (attach-after-demon!
  (lambda ()
   (when (one (lambda (xe) (constraint xe))
              (domain-variable-domain x))
    (restrict-domain!
     x
     (list
      (find-if (lambda (xe) (constraint xe))
               (domain-variable-domain x))))))
  x))
```

Value Propagation in Scheme—II

```
(define (assert-binary-constraint-vp! constraint x y)
 (for-each
  (lambda (v)
   (attach-after-demon!
    (lambda ()
     (when (bound? x)
      (when (one (lambda (ye) (constraint (binding x) ye))
                 (domain-variable-domain y))
       (restrict-domain!
        y (list (find-if (lambda (ye) (constraint (binding x) ye))
                         (domain-variable-domain y))))))
     (when (bound? y)
      (when (one (lambda (xe) (constraint xe (binding y)))
                 (domain-variable-domain x))
       (restrict-domain!
        x (list (find-if (lambda (xe) (constraint xe (binding y)))
                         (domain-variable-domain x)))))))
    v))
  (list x y)))
```

Cascaded Value Propagation



Bounded: Each inference binds a variable

$O(n)$ inferences

Each constraint is checked at most once per argument