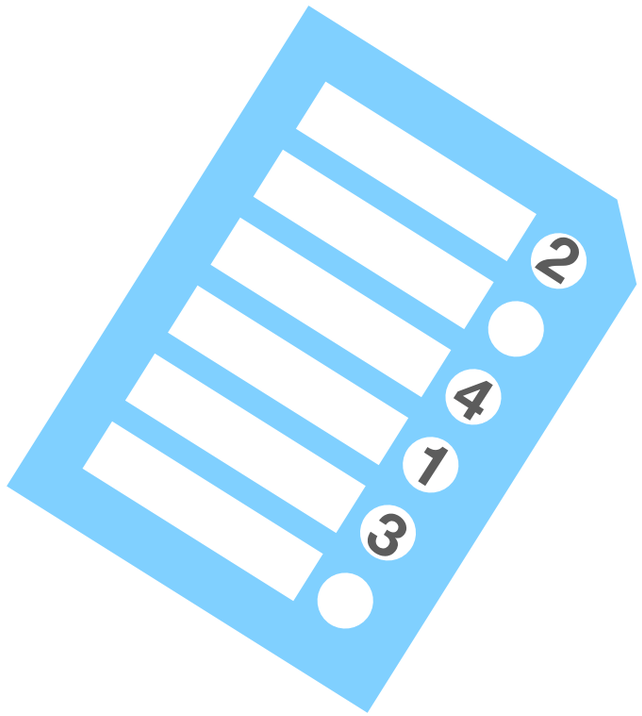
A Formalization of an Instant Run-Off Voting Scheme



Shilpi Goel & Mayank Manjrekar

ACL2-2018 *15 min. Rump Session Talk*

Instant Run-Off Voting

Instant Run-Off Voting

a.k.a. Single-Winner Ranked Choice Voting

Instant Run-Off Voting

a.k.a. Single-Winner Ranked Choice Voting

a.k.a. Single Transferable Voting

Instant Run-Off Voting

a.k.a. Single-Winner Ranked Choice Voting

a.k.a. Single Transferable Voting

a.k.a. Alternative Voting

Instant Run-Off Voting

a.k.a. Single-Winner Ranked Choice Voting

a.k.a. Single Transferable Voting

a.k.a. Alternative Voting

IRV is a *preferential voting* scheme: voters rank candidates in order of preference to elect one winner.

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Where is IRV Used?

- **Politics**
  - US Senate and Congress Race in Maine
  - President of India
  - Mayor of London
  - Members of the Australian Parliament's lower House
- **Entertainment**
  - Oscar's Best Picture Award
- **Computer Science**
  - Planning
  - Rank Aggregation Engines

Where is IRV Used?

- **Politics**
  - US Senate and Congress Race in Maine
  - President of India
  - Mayor of London
  - Members of the Australian Parliament’s lower House
- **Entertainment**
  - Oscar’s Best Picture Award
- **Computer Science**
  - Planning
  - Rank Aggregation Engines

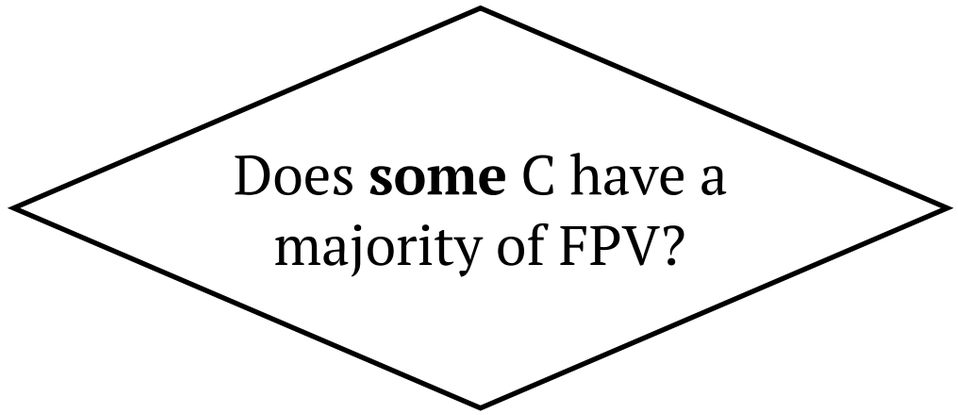
We got interested because of **ACL2-2018's slogan election.**

ACL2-2018: IRV Scheme

C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

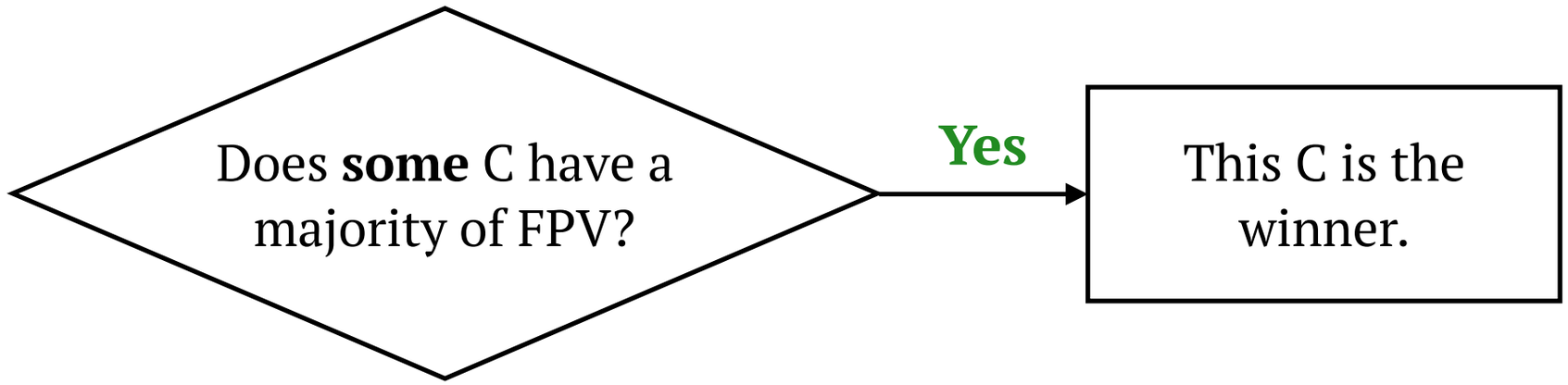
Majority Step



C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

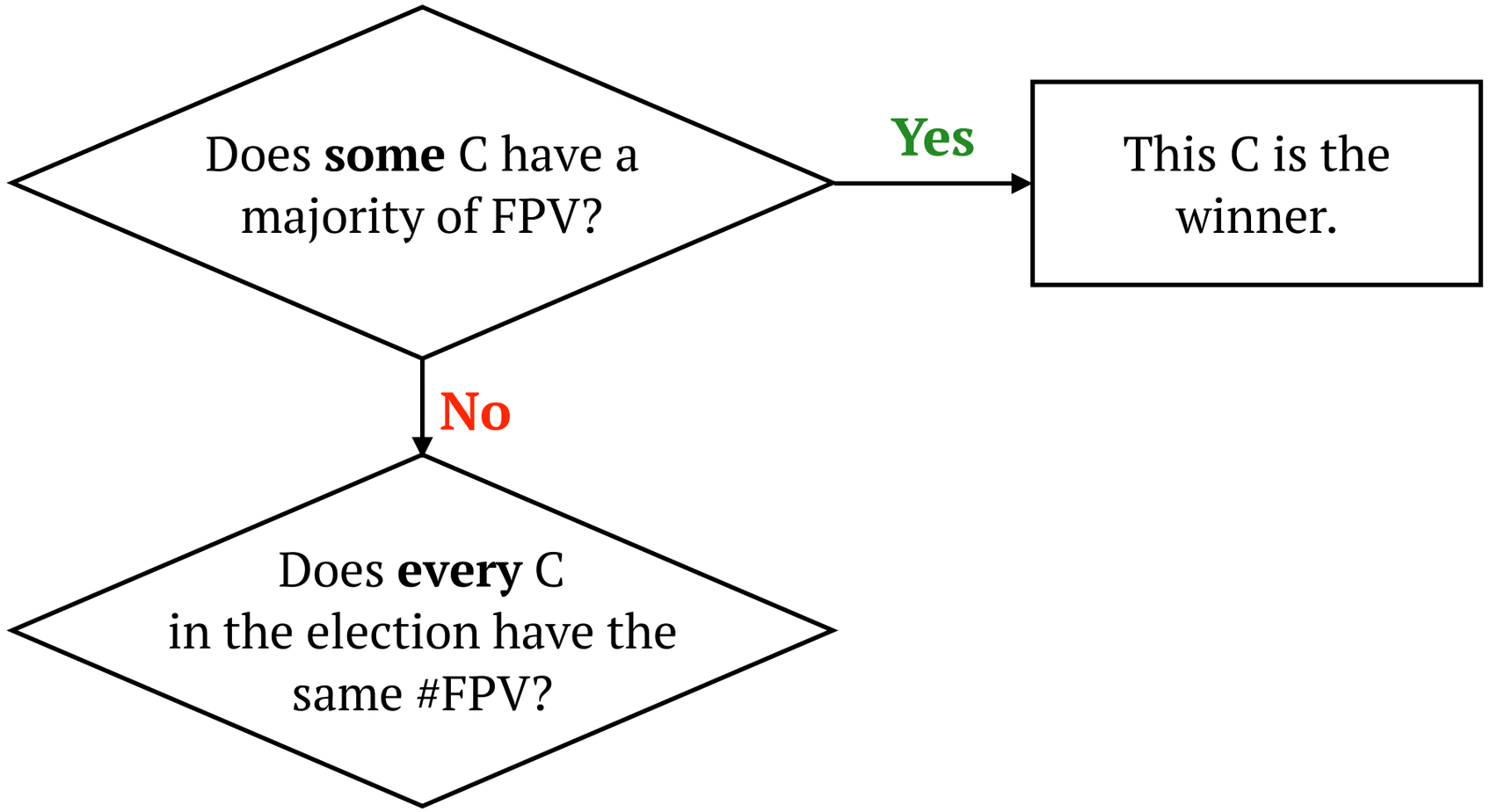
Majority Step



C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

Majority Step

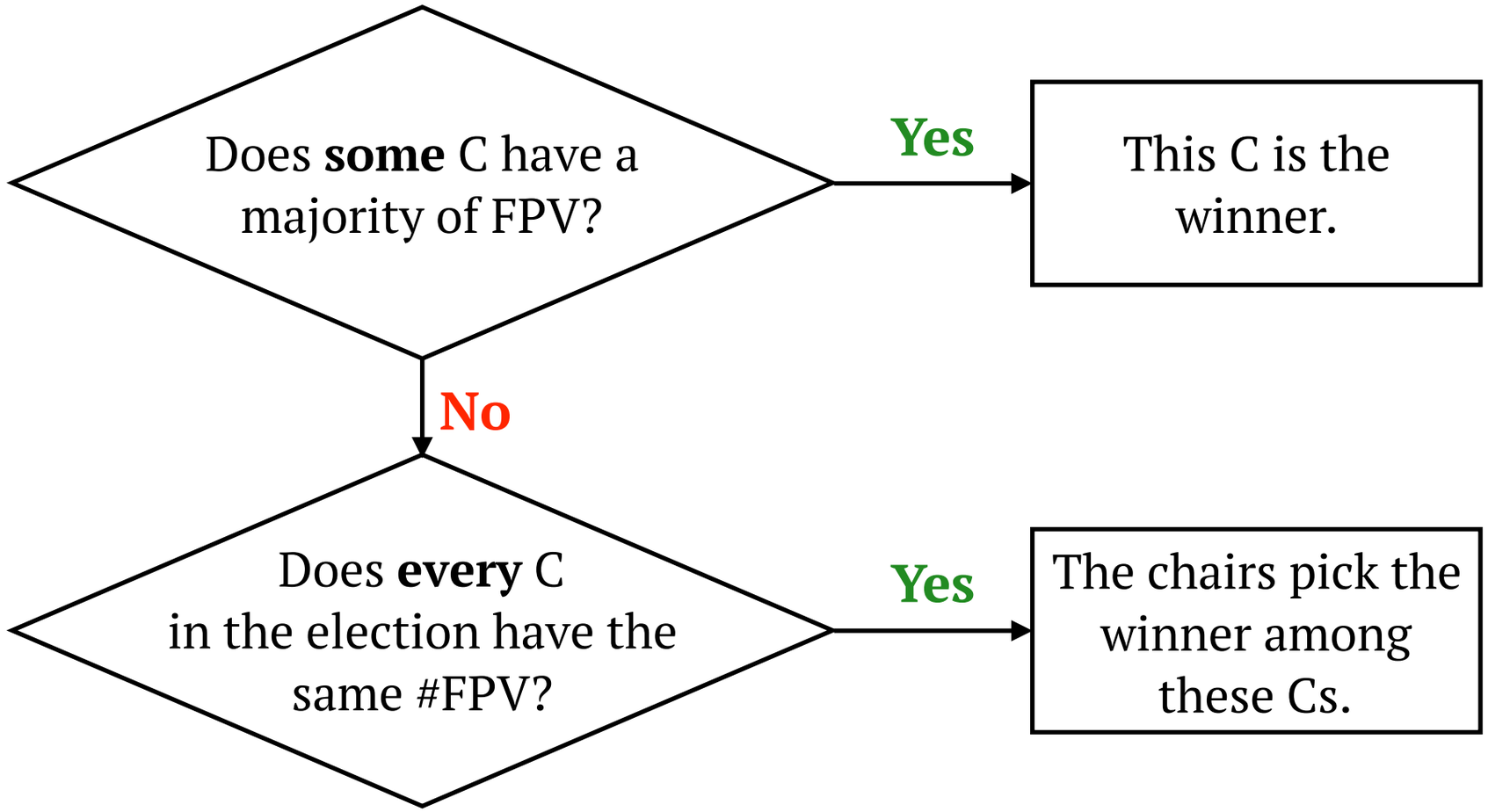


Tie-Breaker Step

C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

Majority Step



Tie-Breaker Step

C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

Majority Step

Does **some** C have a majority of FPV?

Yes → This C is the winner.

No →

Tie-Breaker Step

Does **every** C in the election have the same #FPV?

Yes → The chairs pick the winner among these Cs.

No →

Elimination Step

Eliminate **all** Cs with the least #FPV; **adjust each ballot**.

C	Candidate
FPV	First-Place Votes

ACL2-2018: IRV Scheme

Majority Step

Does **some** C have a majority of FPV?

Yes → This C is the winner.

No →

Tie-Breaker Step

Does **every** C in the election have the same #FPV?

Yes → The chairs pick the winner among these Cs.

No →

Elimination Step

Eliminate **all** Cs with the least #FPV; **adjust each ballot**.

C	Candidate
FPV	First-Place Votes

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

A wins

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

- Vote counting was done manually for ACL2-2018 slogans.

A wins

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

- Vote counting was done manually for ACL2-2018 slogans.
- This scheme seems a little unfair…

A wins

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

- Vote counting was done manually for ACL2-2018 slogans.
- This scheme seems a little unfair…

  …different notions of fairness

A wins

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

A wins

- Vote counting was done manually for ACL2-2018 slogans.
- This scheme seems a little unfair…

  …different notions of fairness
- Matt Kaufmann:

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

	Preference 1	Preference 2	Preference 3
Voter 1	A	-	-
Voter 2	A	-	-
Voter 3	-	-	A
Voter 4	-	-	

	Preference 1
Voter 1	A
Voter 2	A
Voter 3	A

A wins

- Vote counting was done manually for ACL2-2018 slogans.
- This scheme seems a little unfair…

  …different notions of fairness
- Matt Kaufmann:

"if I were to do this again … if there's a tie for least first-place votes, then it's broken by which of those has the least second-place votes, etc., before deleting candidates."

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

	Preference 1	Preference 2
Voter 1	A	C
Voter 2	A	C
Voter 3	C	A
Voter 4	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

	Preference 1	Preference 2
Voter 1	A	C
Voter 2	A	C
Voter 3	C	A
Voter 4	C	

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

	Preference 1	Preference 2
Voter 1	A	C
Voter 2	A	C
Voter 3	C	A
Voter 4	C	

Eliminate A

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

	Preference 1	Preference 2
Voter 1	A	C
Voter 2	A	C
Voter 3	C	A
Voter 4	C	

Eliminate A

	Preference 1
Voter 1	C
Voter 2	C
Voter 3	C
Voter 4	C

[books]/projects/irv

	Preference 1	Preference 2	Preference 3
Voter 1	A	B	C
Voter 2	A	C	B
Voter 3	C	B	A
Voter 4	B	C	

Pick-Candidate(B,C) = B

	Preference 1	Preference 2
Voter 1	A	C
Voter 2	A	C
Voter 3	C	A
Voter 4	C	

Eliminate A

	Preference 1
Voter 1	C
Voter 2	C
Voter 3	C
Voter 4	C

C wins

[books]/projects/irv

[books]/projects/irv

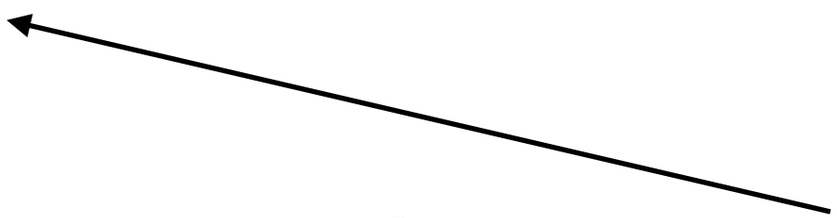
```
(defun irv (xs)
  (cond
   ((or (not (irv-ballot-p xs))
        (endp xs))
    nil)
   (t
    (b* ((cids (candidate-ids xs))
         (maj-winner? (first-choice-of-majority-p cids xs))
         ((when (natp maj-winner?)) maj-winner?)
         (weak-candidate
          (candidate-with-least-nth-place-votes 0 cids xs))
         (new-xs (eliminate-candidate weak-candidate xs)))
      (irv new-xs)))))
```

[books]/projects/irv

```
(defun candidate-with-least-nth-place-votes (n cids xs)
 (cond ((endp cids) nil)

       ((< n (number-of-candidates xs))
        (let* ((relevant-candidates
                (candidates-with-min-votes n cids xs)))
          (if (equal (len relevant-candidates) 1)
              (car relevant-candidates)
            (candidate-with-least-nth-place-votes
              (1+ n) relevant-candidates xs))))

       (t
        ;; Tie persisted throughout all the preference
        ;; levels.  Use a tie-breaker function.
        (pick-candidate cids))))
```

[books]/projects/irv

```
(defun candidate-with-least-nth-place-votes (n cids xs)
 (cond ((endp cids) nil)

       ((< n (number-of-candidates xs))
        (let* ((relevant-candidates
                (candidates-with-min-votes n cids xs)))
          (if (equal (len relevant-candidates) 1)
              (car relevant-candidates)
            (candidate-with-least-nth-place-votes
              (1+ n) relevant-candidates xs))))

       (t
        ;; Tie persisted throughout all the preference
        ;; levels.  Use a tie-breaker function.
        (pick-candidate cids))))
```



Constrained function:
returns a member of its input

Stuff We Don’t Have Time For

- **Social Choice Theory**
  - Very rich
  - Many other possibly “better” schemes
  - Voting schemes can be quite controversial

Stuff We Don’t Have Time For

- **Social Choice Theory**
  - Very rich
  - Many other possibly “better” schemes
  - Voting schemes can be quite controversial
  - ***All*** “reasonable” fairness criteria cannot be satisfied:
    - Arrow’s Impossibility Theorem
    - Gibbard–Satterthwaite Theorem
- **Computational Choice Theory**

Stuff We Don’t Have Time For

- **Social Choice Theory**
  - Very rich
  - Many other possibly “better” schemes
  - Voting schemes can be quite controversial
  - ***All*** “reasonable” fairness criteria cannot be satisfied:
    - Arrow’s Impossibility Theorem
    - Gibbard–Satterthwaite Theorem
- **Computational Choice Theory**
  - Additional sets of concerns: parallelizability, importance of tie-breaking, etc.

Fairness Criteria

Our formalization meets the following criteria that should be satisfied by IRV schemes:

- **Majority Winner Criterion**
- **Condorcet Loser Criterion**
- **Majority Loser Criterion**

Majority Winner Criterion

If a candidate is preferred by an absolute majority of voters, then that candidate must win.

```
(defthm irv-satisfies-the-majority-criterion
  (implies (and (< (majority (number-of-voters xs))
                   (count e (make-nth-choice-list 0 xs)))
                (irv-ballot-p xs))
           (equal (irv xs) e)))
```

Majority Winner Criterion

If a candidate is preferred by an absolute majority of voters, then that candidate must win.

```
(defthm irv-satisfies-the-majority-criterion
  (implies (and (< (majority (number-of-voters xs))
                   (count e (make-nth-choice-list 0 xs)))
                (irv-ballot-p xs))
           (equal (irv xs) e)))
```

Straightforward; needed lemmas like:

- If `e` gets the majority of first-place votes, then there cannot be a tie for the maximum number of first-place votes.

Condorcet Loser Criterion

If a candidate L loses a head-to-head competition against every other candidate, then L must not win the overall election.

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

Condorcet Loser Criterion

If a candidate L loses a head-to-head competition against every other candidate, then L must not win the overall election.

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

Proof Sketch: Let `w = (irv xs)`.

Condorcet Loser Criterion

If a candidate L loses a head-to-head competition against every other candidate, then L must not win the overall election.

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

Proof Sketch: Let `w = (irv xs)`.

- If `w` won by a majority, then `w` would still have majority in every head-to-head competition; therefore, `w != l`.

Condorcet Loser Criterion

If a candidate L loses a head-to-head competition against every other candidate, then L must not win the overall election.

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

Proof Sketch: Let `w = (irv xs)`.

- If `w` won by a majority, then `w` would still have majority in every head-to-head competition; therefore, `w != l`.
- Otherwise, induct on `xs`.

Condorcet Loser Criterion

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

- ***Base Case:*** `xs` has two candidates.

Condorcet Loser Criterion

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

- ***Base Case:*** `xs` has two candidates.
- ***Inductive Step:*** Let the statement be true for `(eliminate-candidate id xs)`,where `id` is picked by `candidate-with-least-nth-place-votes`.

Condorcet Loser Criterion

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

- ***Base Case:*** `xs` has two candidates.
- ***Inductive Step:*** Let the statement be true for `(eliminate-candidate id xs)`, where `id` is picked by `candidate-with-least-nth-place-votes`.
  - Note that `w ≠ id`.

Condorcet Loser Criterion

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

- ***Base Case:*** `xs` has two candidates.
- ***Inductive Step:*** Let the statement be true for `(eliminate-candidate id xs)`, where `id` is picked by `candidate-with-least-nth-place-votes`.
  - Note that `w ≠ id`.
  - If `l == id`, then `w ≠ l`.

Condorcet Loser Criterion

```
(defthm irv-satisfies-the-condorcet-loser-criterion
  (implies (and (all-head-to-head-competition-loser-p l cids xs)
                (set-equiv (cons l cids) (candidate-ids xs))
                (no-duplicatesp-equal (cons l cids))
                (nat-listp cids) (<= 1 (len cids))
                (irv-ballot-p xs))
           (not (equal (irv xs) l))))
```

- ***Base Case:*** `xs` has two candidates.
- ***Inductive Step:*** Let the statement be true for `(eliminate-candidate id xs)`, where `id` is picked by `candidate-with-least-nth-place-votes`.
  - Note that `w ≠ id`.
  - If `l == id`, then `w ≠ l`.
  - Otherwise, `l` is still the head-to-head loser after `id` is eliminated. So by the induction hypothesis, `w ≠ l`.

Majority Loser Criterion

If a majority of voters prefers every other candidate over a given candidate l, then l must not win.

Note that l has a majority of last-place votes.

```
(defthm irv-satisfies-the-majority-loser-criterion
  (implies
   (and (< (majority (number-of-voters xs))
           (count l (make-nth-choice-list (last-place xs) xs)))
        (< 1 (number-of-candidates xs))
        (irv-ballot-p xs))
   (not (equal (irv xs) l))))
```

Majority Loser Criterion

If a majority of voters prefers every other candidate over a given candidate l, then l must not win.

Note that l has a majority of last-place votes.

```
(defthm irv-satisfies-the-majority-loser-criterion
  (implies
(and (< (majority (number-of-voters xs))
        (count l (make-nth-choice-list (last-place xs) xs)))
     (< 1 (number-of-candidates xs))
     (irv-ballot-p xs))
(not (equal (irv xs) l))))
```

But, a candidate who gets the majority of last-place votes must be the Condorcet Loser.

Conclusion

```
:doc irv::instant-runoff-voting
```

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes

Conclusion

```
:doc irv::instant-runoff-voting
```

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:

Conclusion

`:doc irv::instant-runoff-voting`

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?

Conclusion

```
:doc irv::instant-runoff-voting
```

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?
  - Any area where everyone's opinion ought to count:

Conclusion

`:doc irv::instant-runoff-voting`

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?
  - Any area where everyone's opinion ought to count:
    - Picking a dinner place?

Conclusion

`:doc irv::instant-runoff-voting`

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?
  - Any area where everyone's opinion ought to count:
    - Picking a dinner place?
    - Next read for your book club?

Conclusion

`:doc irv::instant-runoff-voting`

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?
  - Any area where everyone's opinion ought to count:
    - Picking a dinner place?
    - Next read for your book club?
    - …

Conclusion

`:doc irv::instant-runoff-voting`

- Fun project for us; taught us a lot about voting schemes
- Possible future work: include other properties & schemes
- Applications:
  - Slogan winner for the next ACL2 Workshop?
  - Any area where everyone's opinion ought to count:
    - Picking a dinner place?
    - Next read for your book club?
    - …

Thanks!