

Towards a miniKanren with fair search strategies

Kuang-Chen Lu

Weixi Ma

Daniel P. Friedman

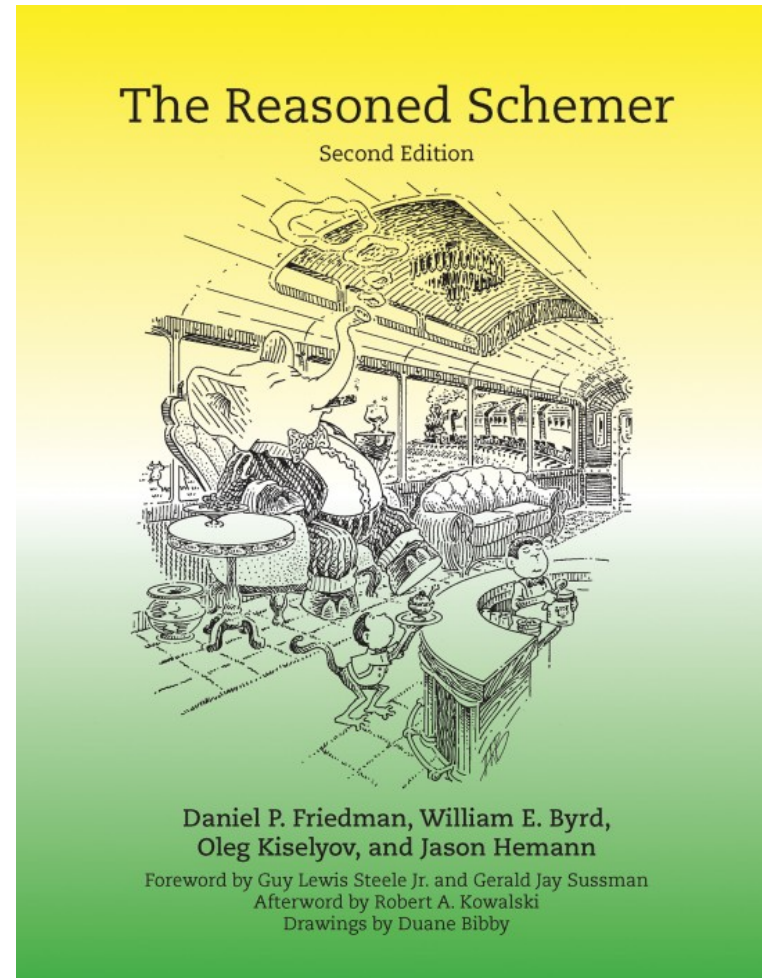
Content

- Background
- Contributions
- The Behavior of Strategies
- The Implementation of Strategies
- Conclusion

Content


- Background ◁
- Contributions
- The Behavior of Strategies
- The Implementation of Strategies
- Conclusion

The Reasoned Schemer, 2nd Edition



Commandments for miniKanren Programmers

- Within each sequences of goals, move non-recursive goals before recursive goals
- If your miniKanren program is slow, fiddle with its **cond^e**-lines.

(interleaving DFS, the strategy of  and μ Kanren, has unfair disjunction)

Cases Where We Might Want Other Strategies

- teaching new miniKanren programmers
- writing relational definitions that run in different running modes

Content

- Background ✓
- Contributions ◁
- The Behavior of Strategies
- The Implementation of Strategies
- Conclusion

Fairness

- fairness in disjunctions: unfair, almost-fair, fair
- fairness in conjunctions: unfair, fair

Search Strategies

strategy	disj	conj
interleaving DFS	unfair	unfair
balanced interleaving DFS	almost-fair	unfair
fair DFS	fair	unfair
BFS[1]	fair	fair

[1] Seres, Silvija, J. Michael Spivey, and C. A. R. Hoare. "Algebra of Logic Programming." ICLP. 1999.

Search Strategies

strategy	disj	conj
interleaving DFS	unfair	unfair
 balanced interleaving DFS	almost-fair	unfair
 fair DFS	fair	unfair
BFS[1]	fair	fair

[1] Seres, Silvija, J. Michael Spivey, and C. A. R. Hoare. "Algebra of Logic Programming." ICLP. 1999.

Content

- Background ✓
- Contributions ✓
- The Behavior of Strategies ◁
- The Implementation of Strategies
- Conclusion

(repeat^o x l)

- **x** is an arbitrary term.
- **l** is a list of one or more **x** s.

`(repeato x l)`

- `x` is an arbitrary term.
- `l` is a list of one or more `x` s.

```
(defrel (repeato x l)
  (conde
    [(== `(,x) l)]
    [(fresh (res)
      (== (cons x res) l)
      (repeato x res))]))
```

`(repeat° x l)`

- `x` is an arbitrary term.
- `l` is a list of one or more `x` s.

```
(defrel (repeat° x l)
  (conde
    [(== `(,x) l)]
    [(fresh (res)
      (== (cons x res) l)
      (repeat° x res))]))
```

```
> (run 3 q
  (repeat° 'λ q))

'((λ) (λ λ) (λ λ λ))
```

Interleaving DFS (Unfair Disjunction)

```
> (run 15 q
  (conde
    [(repeato 'λ q)]
    [(repeato '🐑 q)]
    [(repeato '🐏 q)]
    [(repeato '🐱 q)]
    [(repeato '🐱 q)])))
```

```
' ((λ) (λ λ) (🐑) (λ λ λ) (λ λ λ λ)
   (🐑 🐑) (λ λ λ λ λ) (🐏)
   (λ λ λ λ λ λ) (🐑 🐑 🐑)
   (λ λ λ λ λ λ λ) (λ λ λ λ λ λ λ λ)
   (🐑 🐑 🐑 🐑) (λ λ λ λ λ λ λ λ λ)
   (🐏 🐏))
```

Balanced-interleaving DFS (Almost-fair Disjunction)

```
> (run 15 q
  (conde
    [(repeato 'λ q)]
    [(repeato '🐑 q)]
    [(repeato '🐏 q)]
    [(repeato '🐱 q)]
    [(repeato '🐱 q)])))
```

```
' ((🐑) (🐏) (🐱) (λ)
   (🐑 🐑) (🐏 🐏) (🐱 🐱) (🐱)
   (🐑 🐑 🐑) (🐏 🐏 🐏) (🐱 🐱 🐱) (λ λ)
   (🐑 🐑 🐑 🐑) (🐏 🐏 🐏 🐏) (🐱 🐱 🐱 🐱))
```


Fair DFS & BFS (Fair Disjunction)

```
> (run 15 q
  (conde
    [(repeato 'λ q)]
    [(repeato '🐑 q)]
    [(repeato '🐏 q)]
    [(repeato '🐱 q)]
    [(repeato '🐱 q)])))
```

```
' ((λ) (🐑) (🐏)
  (🐱) (🐱)
  (λ λ) (🐑 🐑) (🐏 🐏)
  (🐱 🐱) (🐱 🐱)
  (λ λ λ) (🐑 🐑 🐑) (🐏 🐏 🐏)
  (🐱 🐱 🐱) (🐱 🐱 🐱)
  (🐏 🐏)
  (🐏 🐏))
```

Content

- Background ✓
- Contributions ✓
- The Behavior of Strategies ✓
- The Implementation of Strategies ◁
- Conclusion

Review

- a **goal** is a function from a **state** to a **space**
- a **state** is a way to satisfy some relations
- a (search) **space** is a collection of **state** s

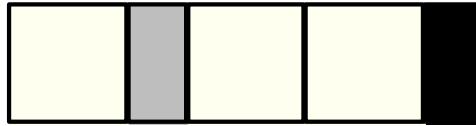
(Search) Space

Space ::= Null | (\rightarrow Space) | (Pair State Space)

(Search) Space

Space ::= Null | (\rightarrow Space) | (Pair State Space)

a space with three states



(Search) Space

Space ::= Null | (\rightarrow Space) | (Pair State Space)

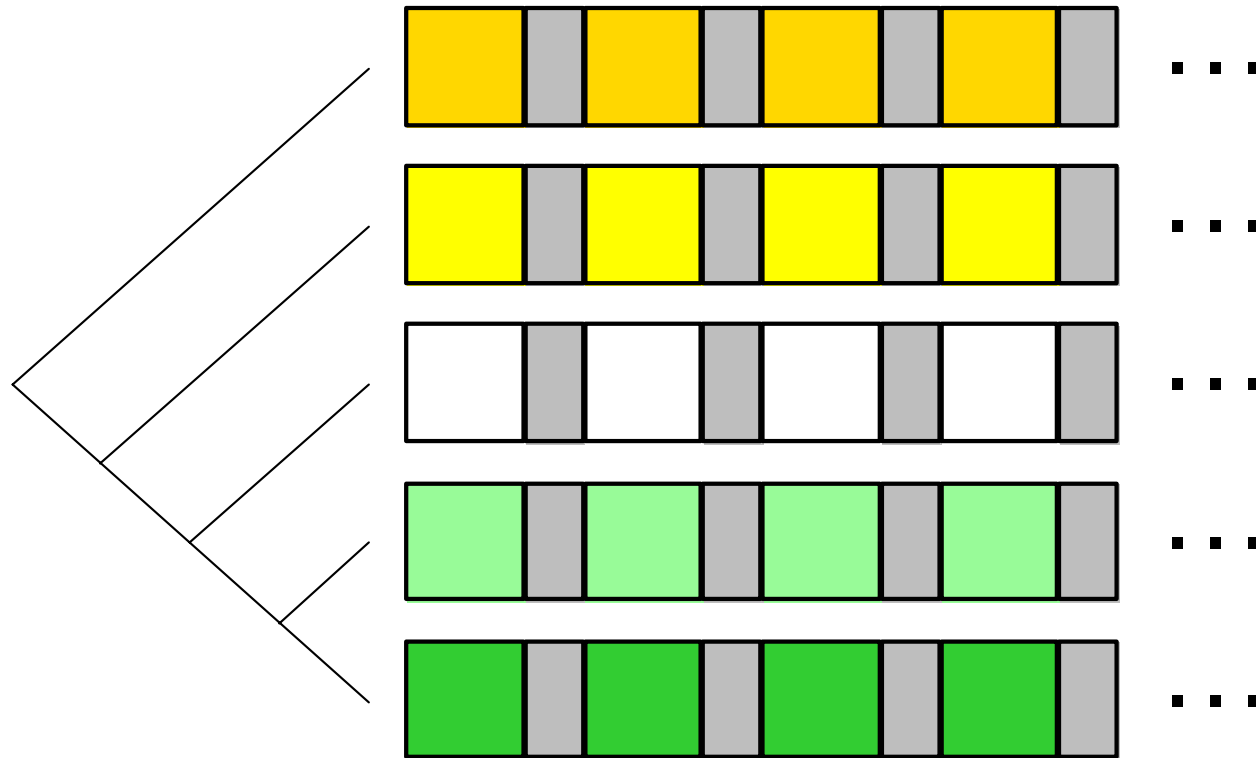
a space with three states



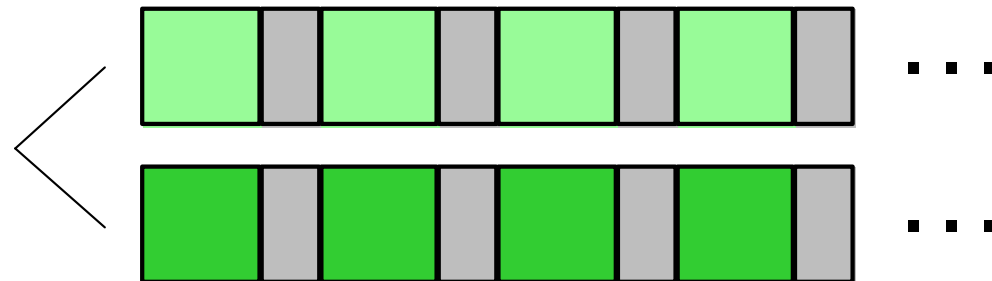
a space with possibly infinite states



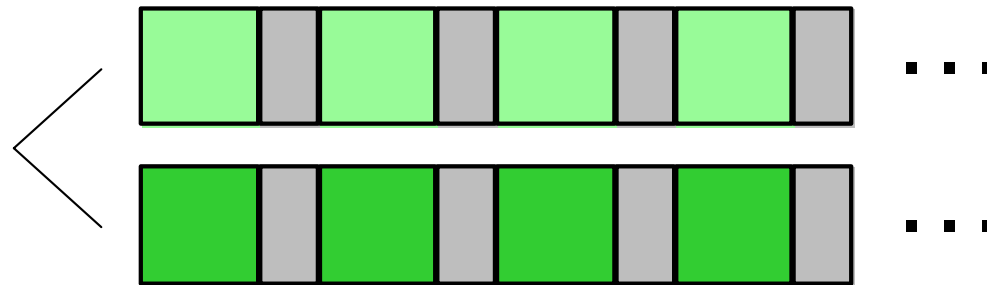
Implementation of Unfair Disjunction



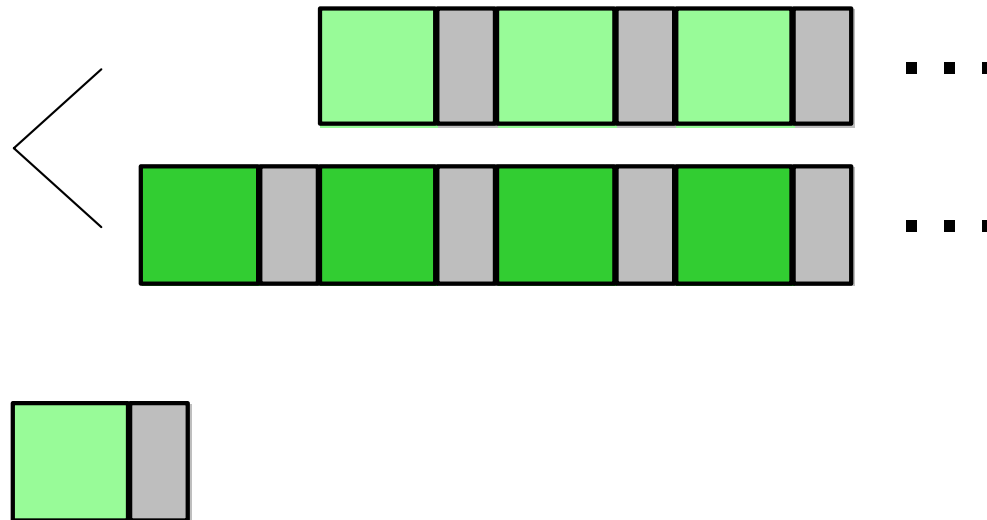
Implementation of Unfair Disjunction



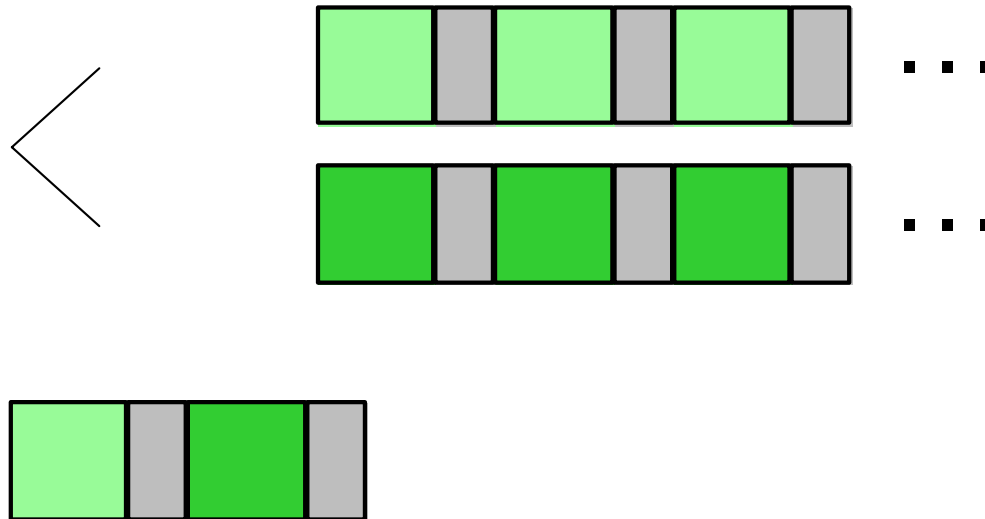
Implementation of Unfair Disjunction



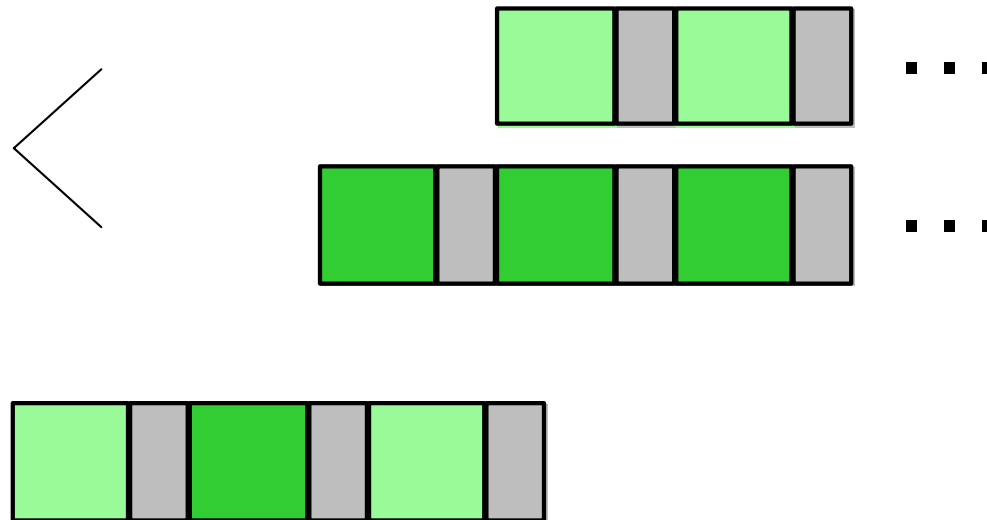
Implementation of Unfair Disjunction



Implementation of Unfair Disjunction



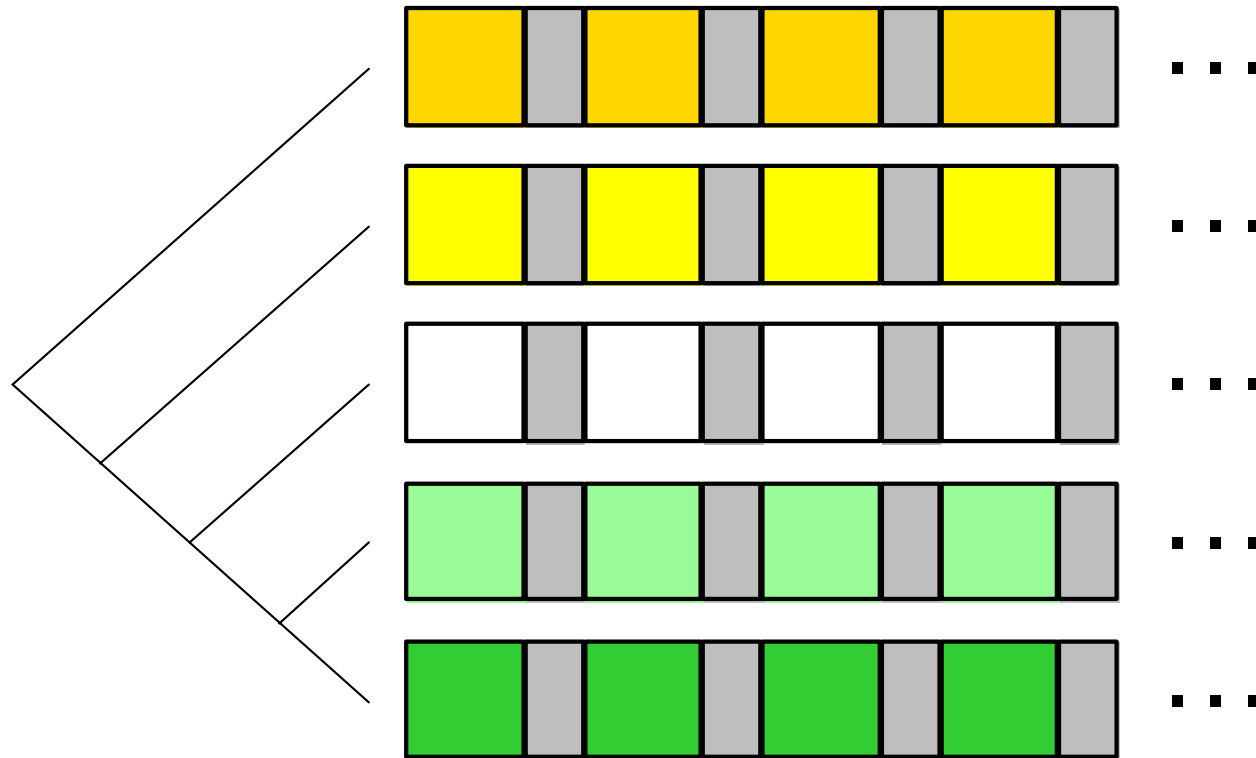
Implementation of Unfair Disjunction



Implementation of Unfair Disjunction

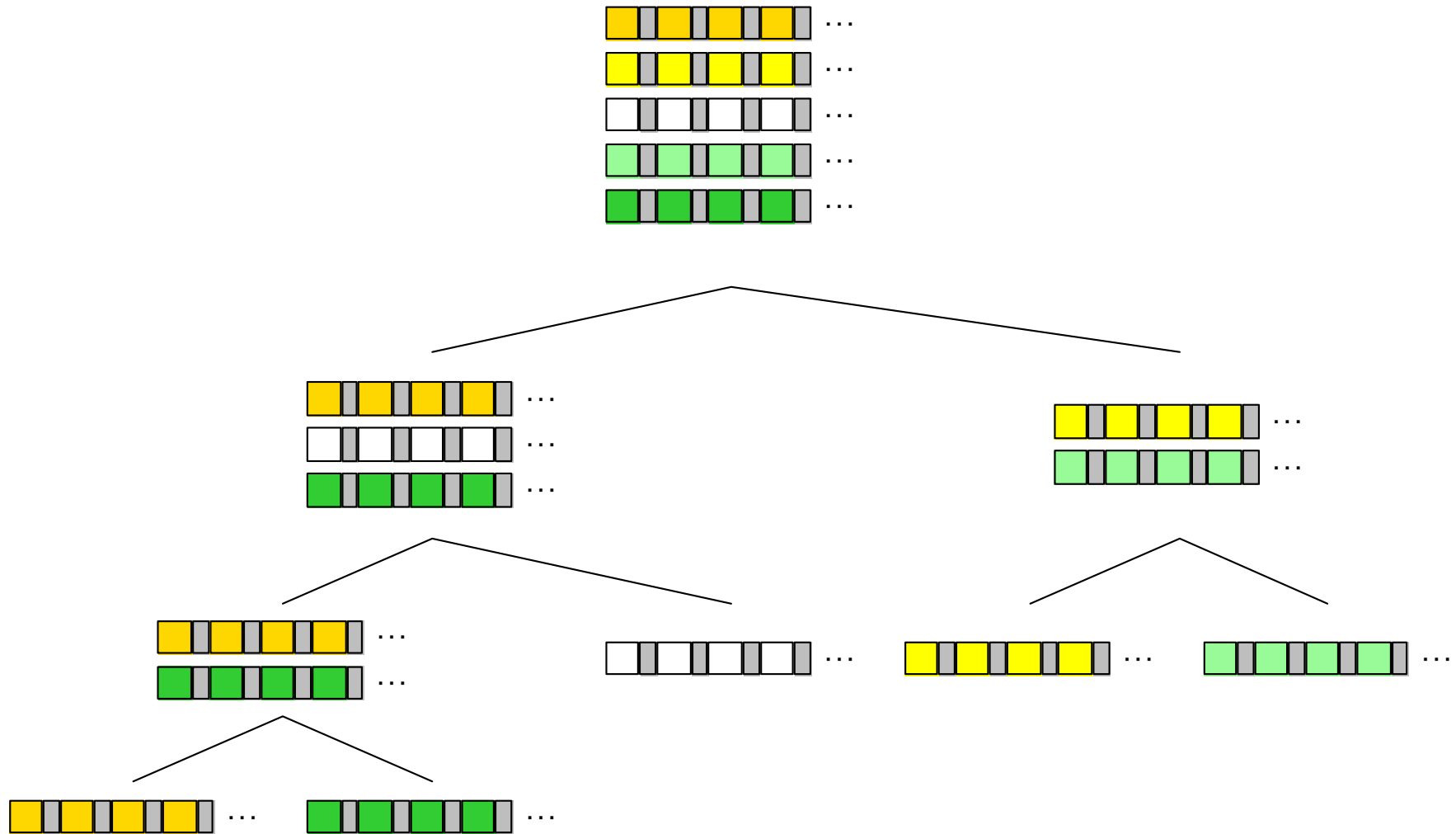


Implementation of Unfair Disjunction



```
(define (append-inf s-inf t-inf)
  (cond
    ((null? s-inf) t-inf)
    ((pair? s-inf)
     (cons (car s-inf)
            (append-inf (cdr s-inf) t-inf)))
    (else (lambda ()
              (append-inf t-inf (s-inf))))))
```

Implementation of Almost-fair Disjunction

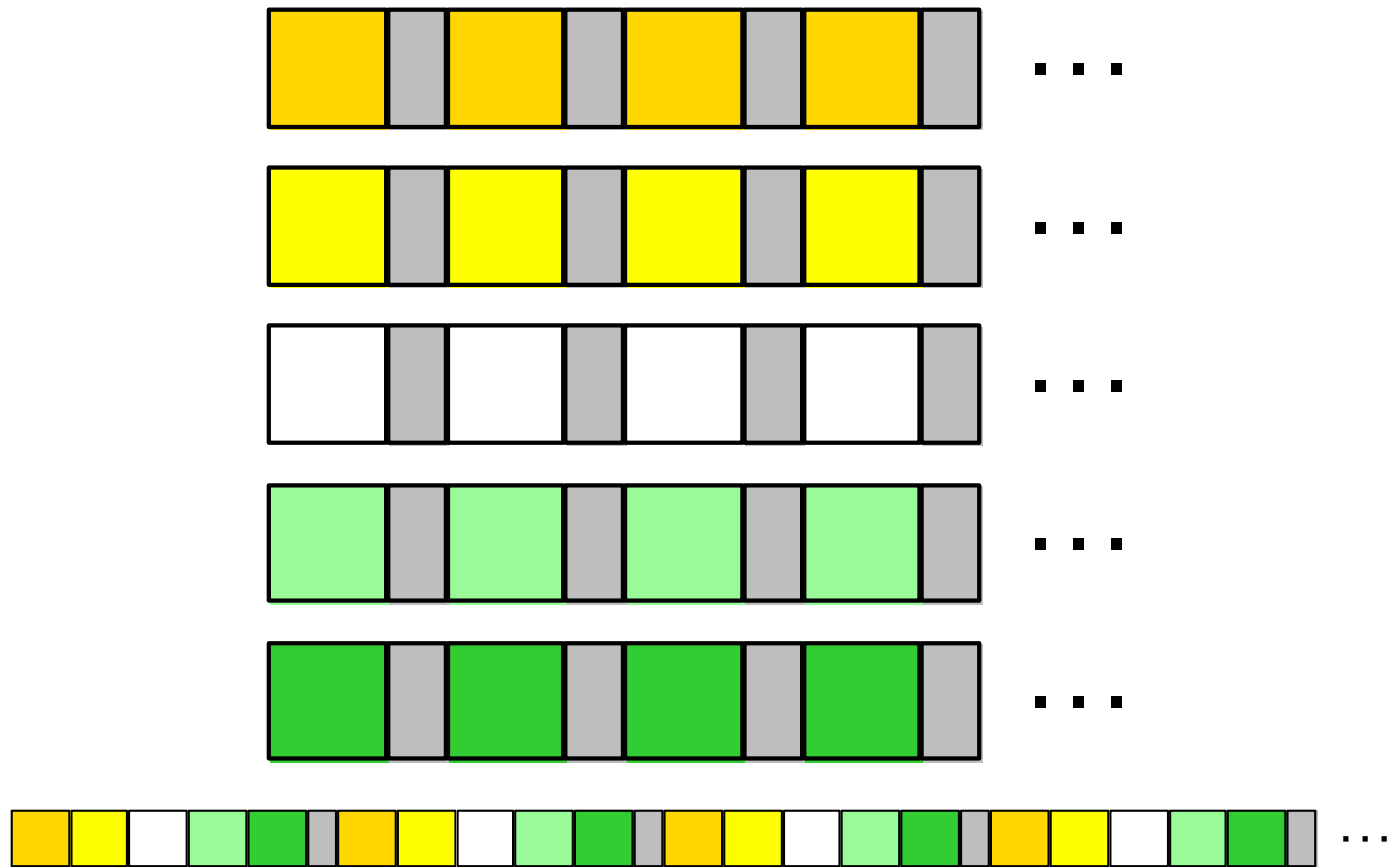


Balanced-interleaving DFS (Almost-fair Disjunction)

```
> (run 15 q
  (conde
    [(repeato 'λ q)]
    [(repeato '🐑 q)]
    [(repeato '🐏 q)]
    [(repeato '🐱 q)]
    [(repeato '🐱 q)])))
```

```
' ((🐑) (🐏) (🐱) (λ)
   (🐑 🐑) (🐏 🐏) (🐱 🐱) (🐱)
   (🐑 🐑 🐑) (🐏 🐏 🐏) (🐱 🐱 🐱) (λ λ)
   (🐑 🐑 🐑 🐑) (🐏 🐏 🐏 🐏) (🐱 🐱 🐱 🐱))
```

Implementation of Fair Disjunction



```
(define (append-inf/fair s-inf t-inf)
  (let loop ((s? #t) (s-inf s-inf) (t-inf t-inf))
    (cond
      ((null? s-inf) t-inf)
      ((pair? s-inf)
       (cons (car s-inf)
              (loop s? (cdr s-inf) t-inf)))
      (s? (loop #f t-inf s-inf))
      (else (lambda ()
                (loop #t (t-inf) (s-inf)))))))
```

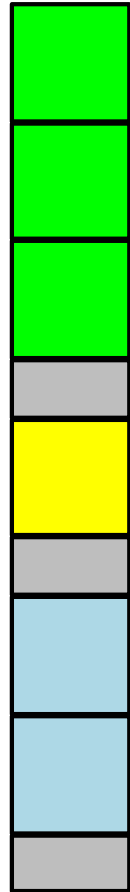
```
(define (append-inf/fair s-inf t-inf)
  (let loop ((s? #t) (s-inf s-inf) (t-inf t-inf))
    (cond
      ((null? s-inf) t-inf)
      ((pair? s-inf)
       (cons (car s-inf)
              (loop s? (cdr s-inf) t-inf)))
      (s? (loop #f t-inf s-inf))
      (else (lambda ()
                (loop #t (t-inf) (s-inf)))))))
```

```
(define (append-inf/fair s-inf t-inf)
  (let loop ((s? #t) (s-inf s-inf) (t-inf t-inf))
    (cond
      ((null? s-inf) t-inf)
      ((pair? s-inf)
       (cons (car s-inf)
              (loop s? (cdr s-inf) t-inf)))
      (s? (loop #f t-inf s-inf))
      (else (lambda ()
                (loop #t (t-inf) (s-inf)))))))
```

Implementations of Conjunctions

Implementations of Conjunctions

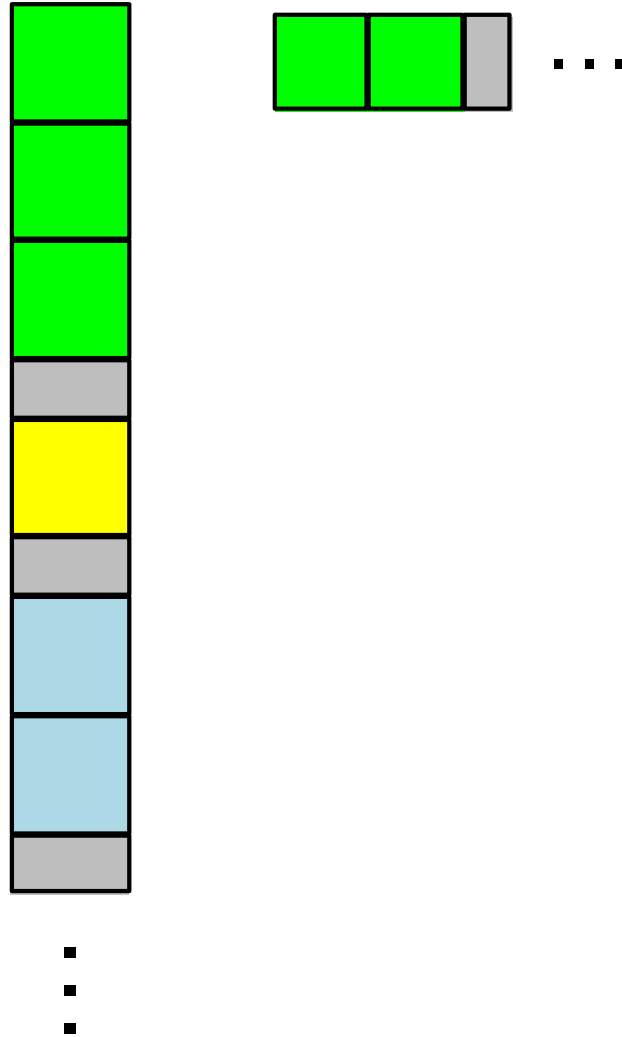
g1 **(conj2 g1 g2)**



⋮

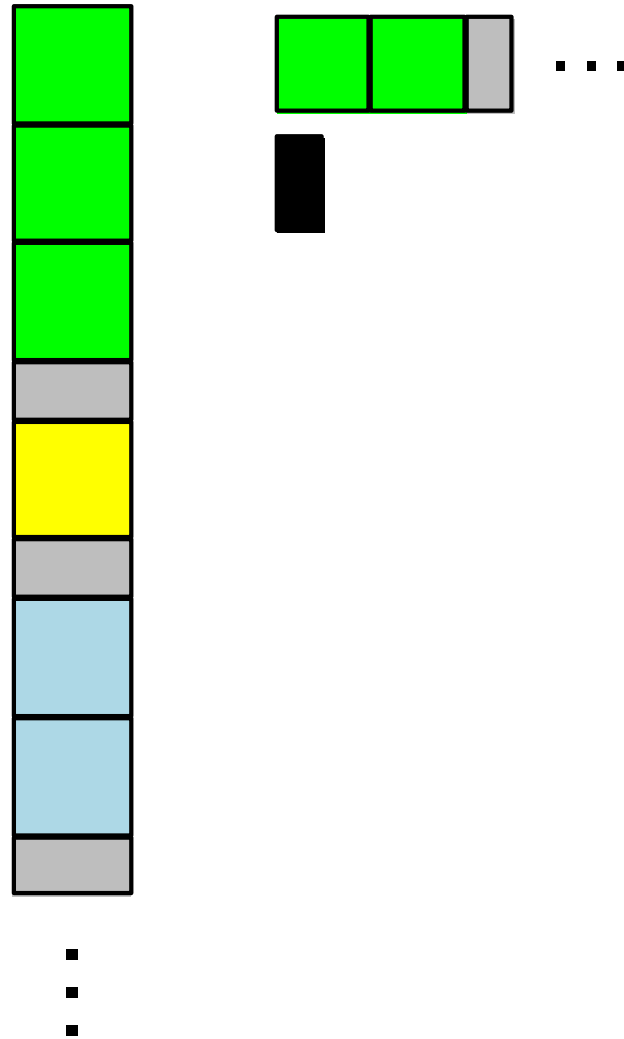
Implementations of Conjunctions

g1 **(conj2 g1 g2)**



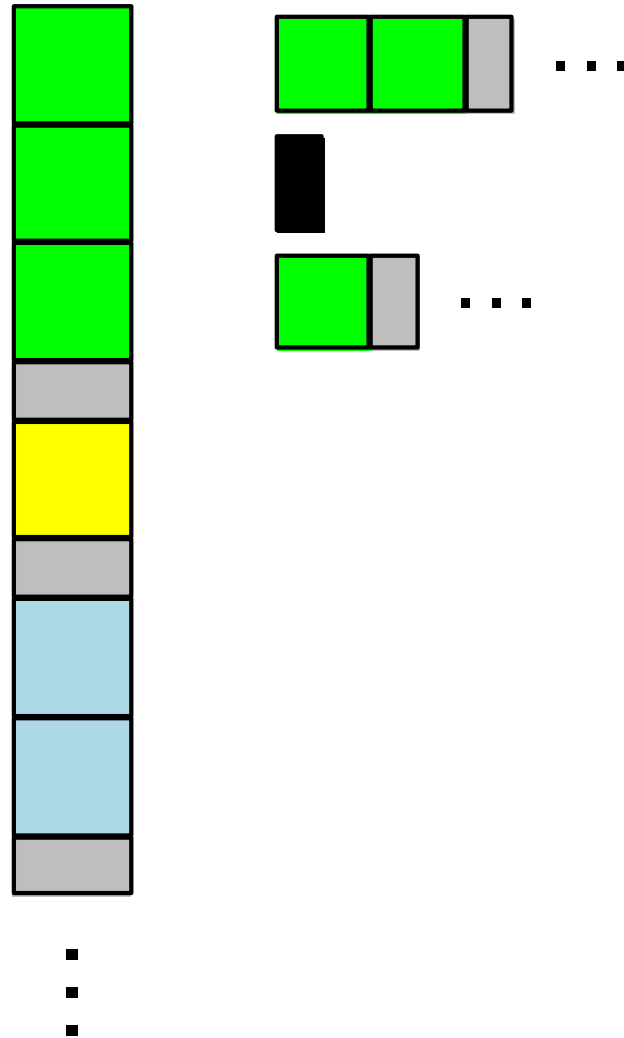
Implementations of Conjunctions

g1 **(conj2 g1 g2)**



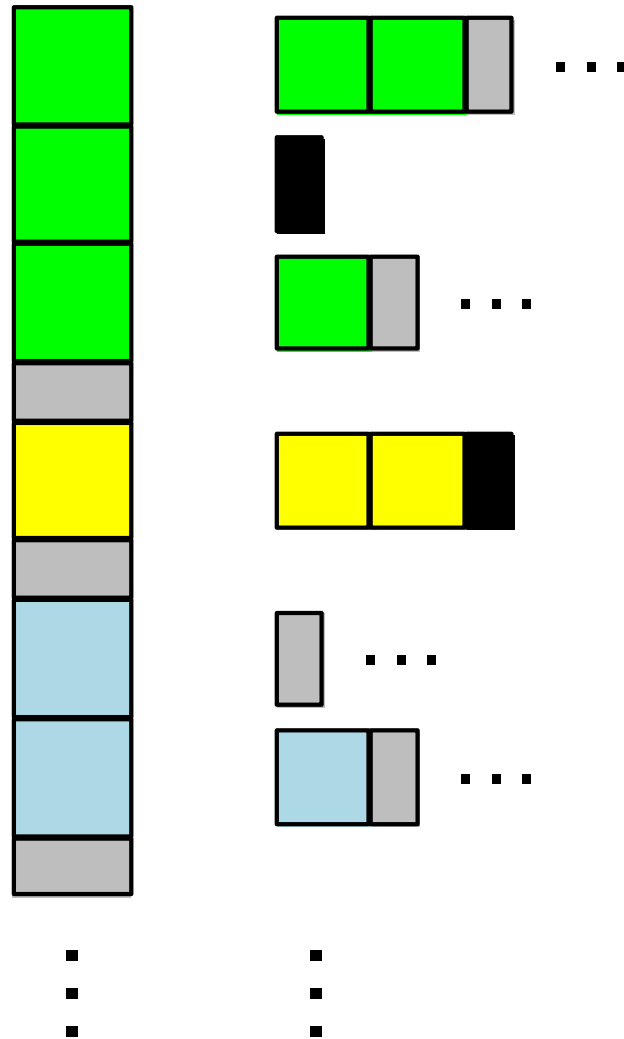
Implementations of Conjunctions

g1 **(conj2 g1 g2)**



Implementations of Conjunctions

g1 **(conj2 g1 g2)**



```
(define (append-map-inf g s-inf)
  (cond
    ((null? s-inf) '())
    ((pair? s-inf)
     (append-inf (g (car s-inf))
                  (append-map-inf g (cdr s-inf))))
    (else (lambda ()
              (append-map-inf g (s-inf))))))
```

```
(define (append-map-inf g s-inf)
  (cond
    ((null? s-inf) '())
    ((pair? s-inf)
     (append-inf (g (car s-inf))
                  (append-map-inf g (cdr s-inf))))
    (else (lambda ()
              (append-map-inf g (s-inf)))))))
```

Content

- Background ✓
- Contributions ✓
- The Behavior of Strategies ✓
- The Implementation of Strategies ✓
- Conclusion ◁

Quantitative Evaluation

Benchmark	DFS_i	DFS_{bi}	DFS_f	BFS_{imp}	BFS_{ser}
quine-1	41	48	33	-	-
quine-2	72	76	32	-	-
'(I love you)-1	78	70	59	254	605
'(I love you)-2	631	173	61	253	605

Why Fairness?

- resistant to permutation of **cond^e** lines.
 - need not be concerned about line order.
 - one definition for different running modes
- more understandable order of answers

Q & A