Automatic Verification for Interactive Graphical Programs

Carl Eastlund
cce@ccs.neu.edu

Matthias Felleisen
matthias@ccs.neu.edu

Northeastern University
Boston, Massachusetts

Verification for I/O and Interactive Programs

Davis. Reasoning about ACL2 file input. ACL2 '06.

Dowse et al. Reasoning about deterministic concurrent functional I/O. IFL '04.

Dwyer et al. Analyzing interaction orderings with model checking. ASE '04.

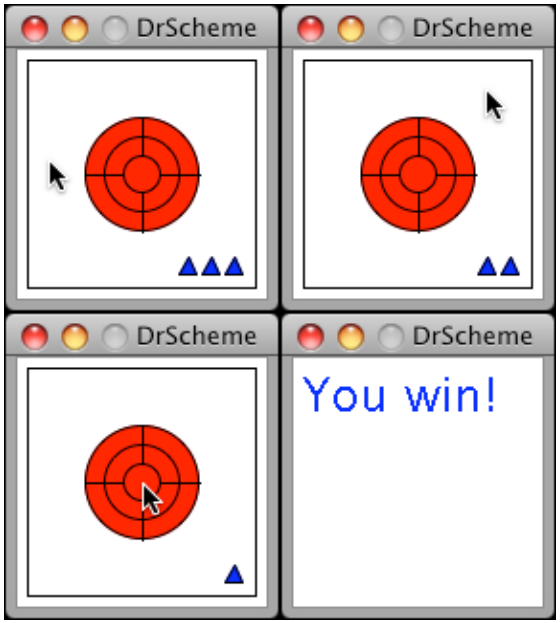
Krishnamurthi and Licata. Verifying interactive web programs. ASE '04.

Godefroid et al. VeriWeb: automatically testing dynamic web sites. WWW '02.

Memon. An event-flow model of GUI-based applications for testing. STVR '07.

Creating Worlds

Dart Game



Dart Game

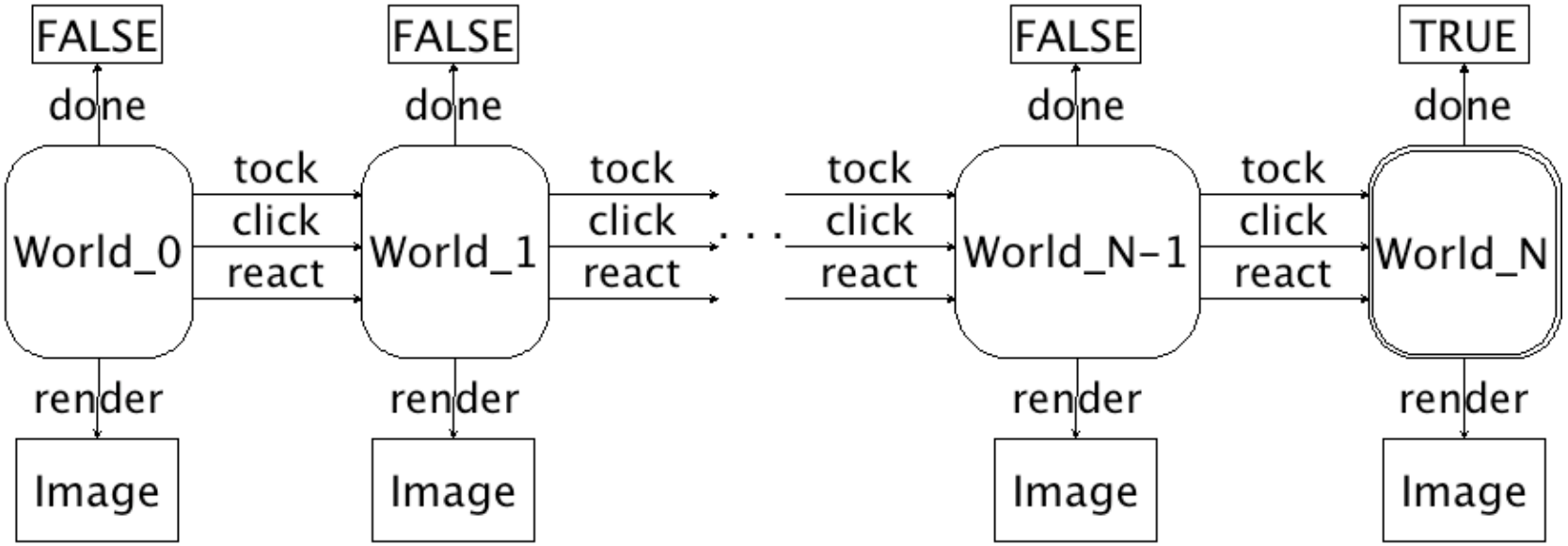
```
; A World is either a Natural Number or 'win

(big-bang 3 ; : World
  (on-draw show-game 600 600)
  (on-mouse throw-dart)
  (stop-when win-or-lose))
```

Dart Game

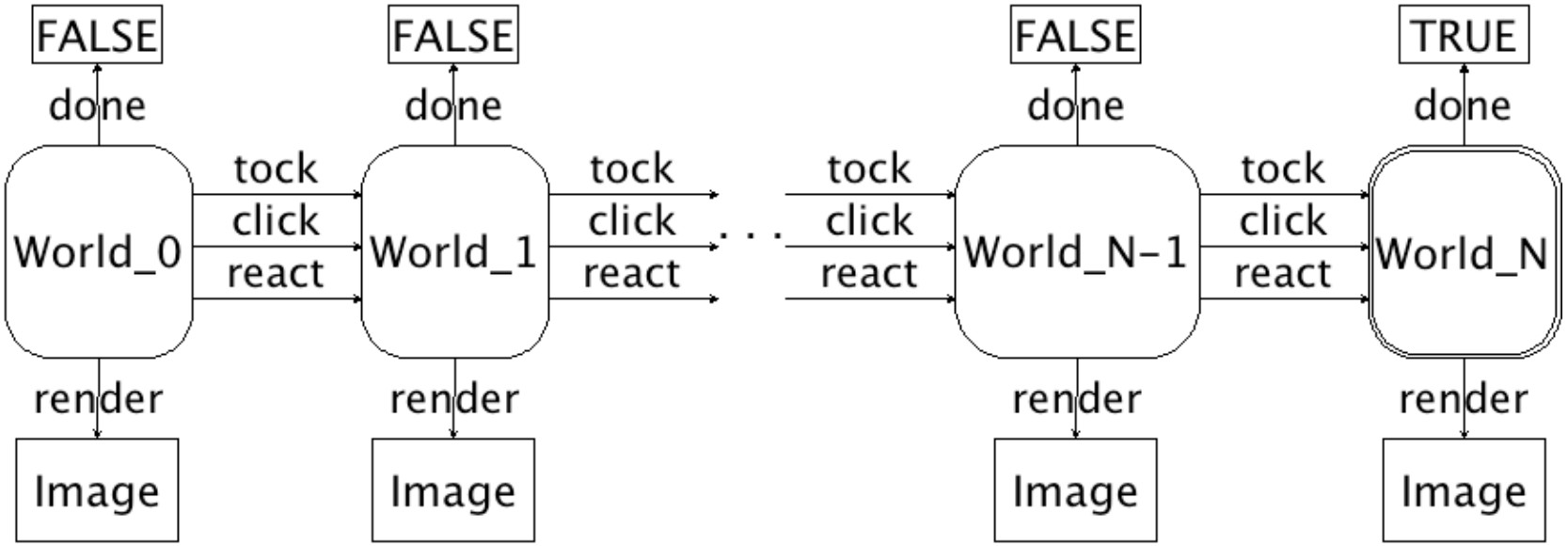
```
; show-game : World -> Image
(defun show-game (w)
  (cond ((equal w 'win) (text "You win!" 120 'blue))
        ((equal w 0) (text "You lose." 120 'blue))
        (t (show-darts w (show-target)))))

; throw-dart : ActiveWorld Int Int -> World
(defun throw-dart (w x y a)
  (if (equal a 'button-down)
      (if (dart-hits x y) 'win (1- w))
      w))

; win-or-lose : World -> Boolean
(defun win-or-lose (w)
  (or (equal w 'win) (equal w 0)))
```

The World Machine

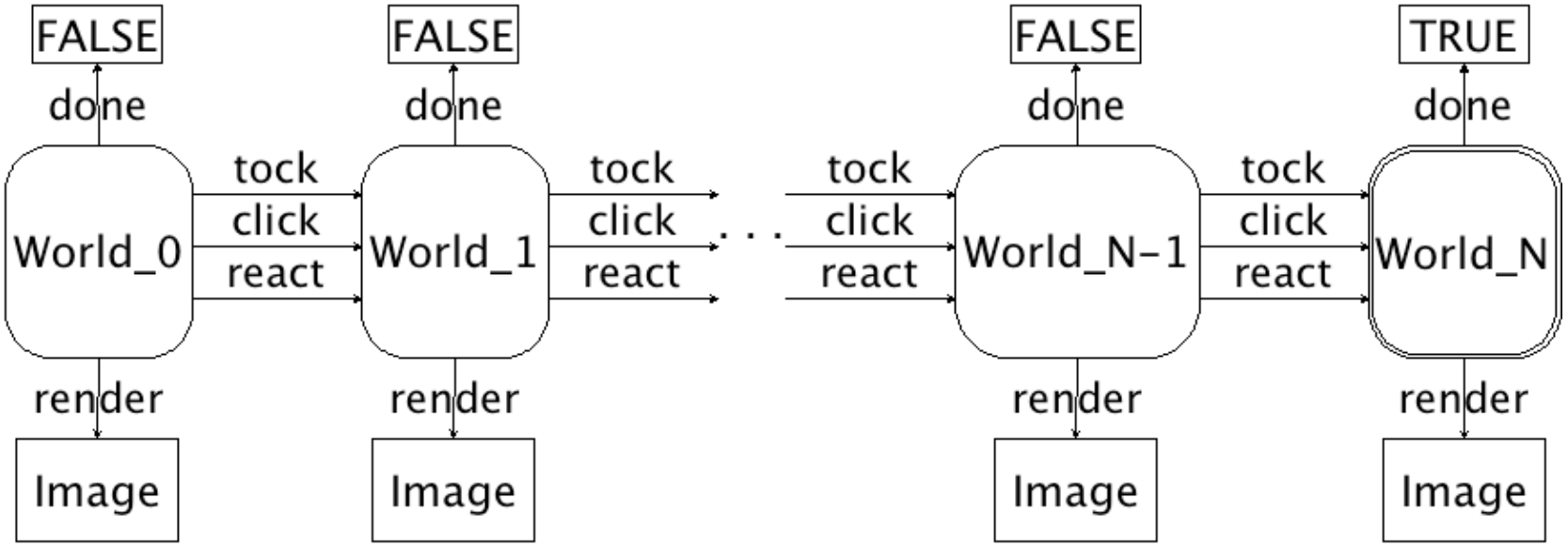
```
(big-bang *WORLD_0*
  (on-draw RENDER *WIDTH* *HEIGHT*)
  (on-tick TOCK *RATE*)
  (on-key REACT)
  (on-mouse CLICK)
  (stop-when DONE))
```

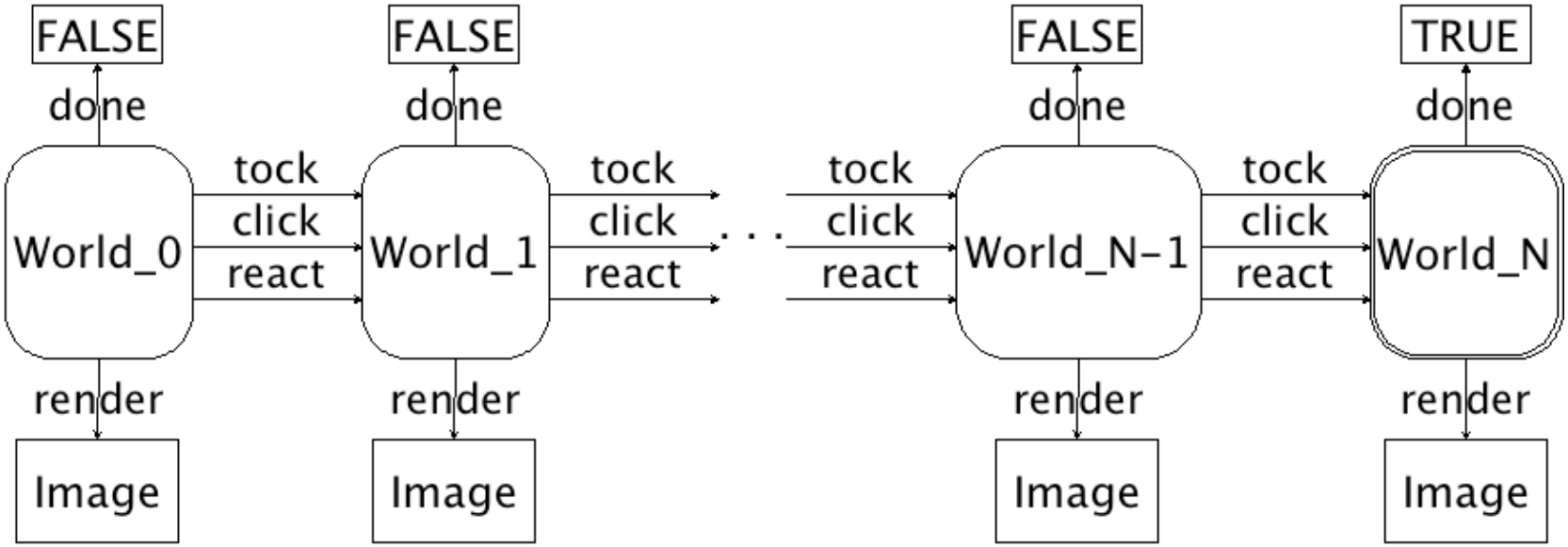


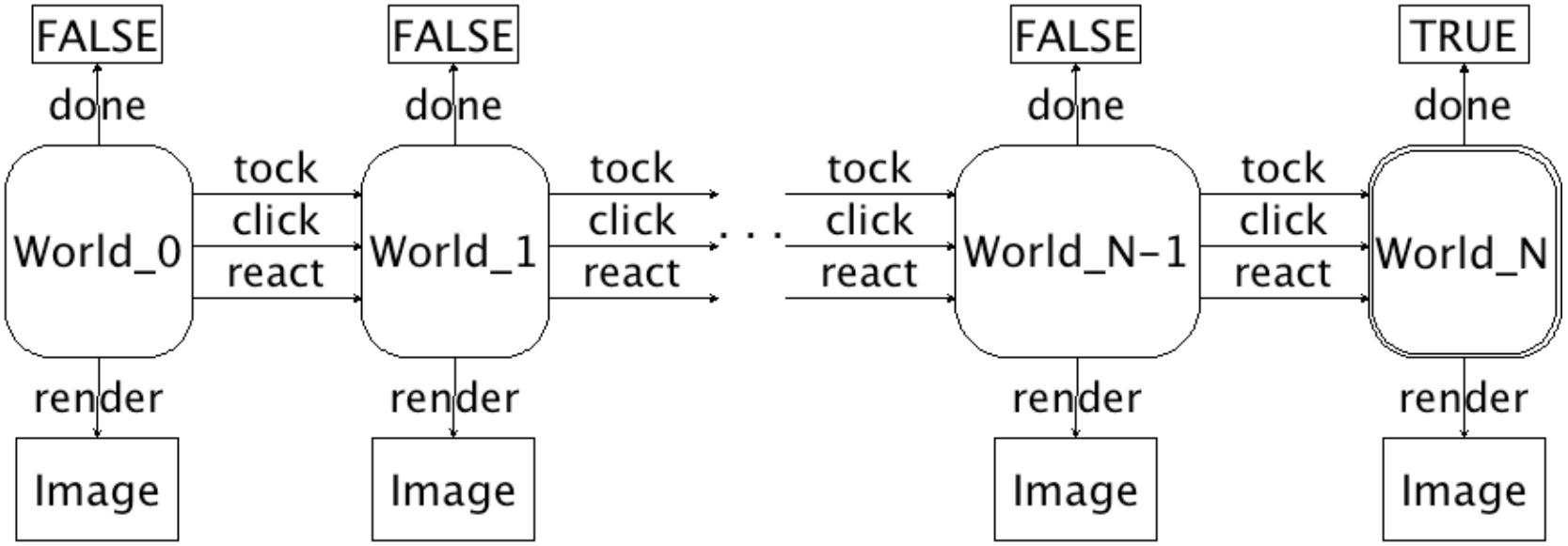
```
(on-draw RENDER *WIDTH* *HEIGHT*)
; RENDER : World -> Image
; *WIDTH*, *HEIGHT* : Nat

(stop-when DONE)
; DONE : World -> Boolean
```



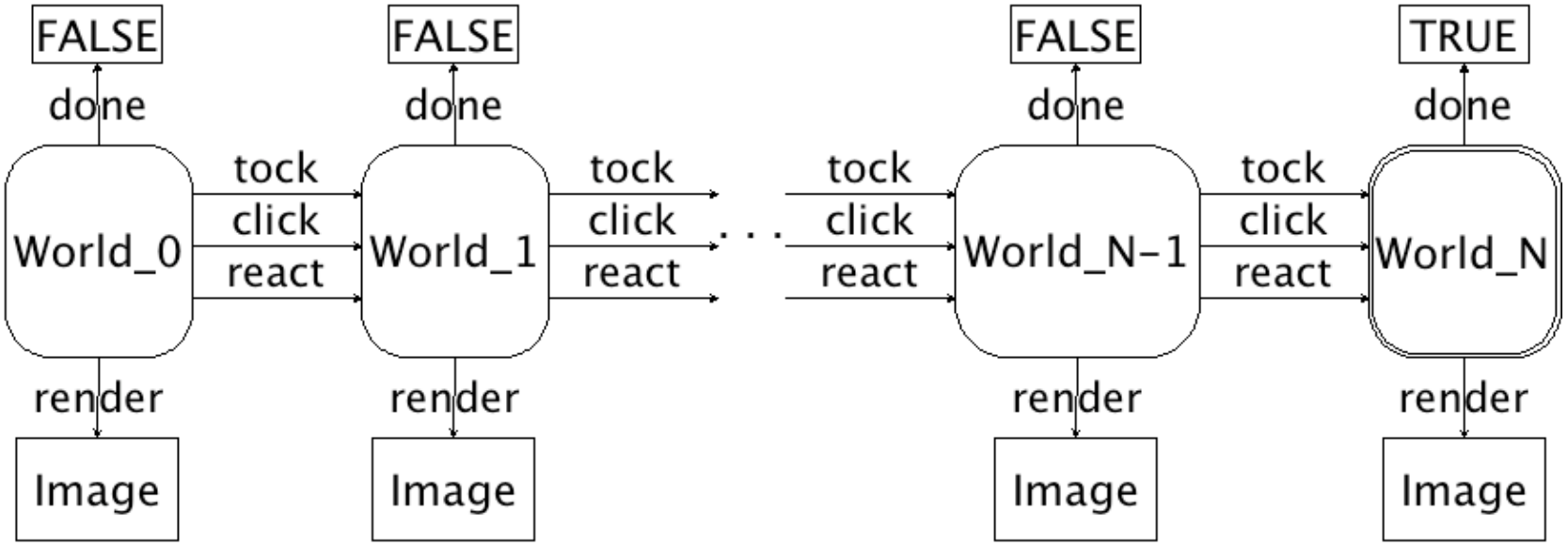
```
(on-tick TOCK *RATE*)
; TOCK : ActiveWorld -> World
; *RATE* : Rational

(on-key REACT)
; REACT : ActiveWorld String -> World

(on-mouse CLICK)
; CLICK : ActiveWorld Int Int Symbol -> World
```




FALSE
FALSE
FALSE
TRUE
done
done
done
done
tock
click
react
tock
click
react
tock
click
react
tock
click
react
World_0
World_1
. . .
World_N-1
World_N
render
render
render
render
Image
Image
Image
Image




```
; event-loop : World EventList -> World
(defun event-loop (w es)
  (cond
   ((endp es) w)
   ((DONE w) w)
   (t (event-loop (event-handler w (car es))
                  (cdr es)))))
```




```
; event-handler : ActiveWorld Event -> World
(defun event-handler (w e)
  (cond
   ((tickp e) (TOCK w))
   ((keyp e) (REACT w e))
   ((mousep e) (CLICK w (mouse-x e)
                        (mouse-y e)
                        (mouse-action e)))
   (t w)))
```

Modeling Darts

```
; event-handler : ActiveWorld Event -> World
(defun event-handler (w e)
  (cond
   ((mousep e) (throw-dart w (mouse-x e)
                             (mouse-y e)
                             (mouse-action e)))
   (t w)))

; event-loop : ActiveWorld EventList -> World
(defun event-loop (w es)
  (cond
   ((endp es) w)
   ((win-or-lose w) w)
   (t (event-loop (event-handler w (car es))
                  (cdr es)))))
```

No Cheating!

No Cheating!

```
(defthm big-bang-darts-left
  (implies (>= (count-clicks es) 3)
           (win-or-lose (event-loop 3 es))))
```

No Cheating!

```
; count-clicks : EventList -> Nat
(defun count-clicks (es)
  (cond ((endp es) 0)
        ((clickp (car es))
         (1+ (count-clicks (cdr es))))
        (t (count-clicks (cdr es)))))

(defthm big-bang-darts-left
  (implies (>= (count-clicks es) 3)
           (win-or-lose (event-loop 3 es))))
```

No Cheating!

```
; clickp : Event -> Boolean
(defun clickp (e)
  (and (mousep e)
       (equal (mouse-action e) 'button-down)))

; count-clicks : EventList -> Nat
(defun count-clicks (es)
  (cond ((endp es) 0)
        ((clickp (car es))
         (1+ (count-clicks (cdr es))))
        (t (count-clicks (cdr es)))))

(defthm big-bang-darts-left
  (implies (>= (count-clicks es) 3)
           (win-or-lose (event-loop 3 es))))
```

No Cheating!

```
(defthm event-loop-darts-left
  (implies (and (dart-gamep w)
                (>= (count-clicks es)
                    (darts-left w)))
           (win-or-lose (event-loop w es))))
```

No Cheating!

```
; darts-left : World -> Nat
(defun darts-left (w)
  (if (natp w) w 0))

(defthm event-loop-darts-left
  (implies (and (dart-gamep w)
                (>= (count-clicks es)
                    (darts-left w)))
           (win-or-lose (event-loop w es))))
```

No Cheating!

```
; dart-gamep : Any -> Boolean
(defun dart-gamep (w)
  (or (natp w) (equal w 'win)))

; darts-left : World -> Nat
(defun darts-left (w)
  (if (natp w) w 0))

(defthm event-loop-darts-left
  (implies (and (dart-gamep w)
                (>= (count-clicks es)
                    (darts-left w)))
           (win-or-lose (event-loop w es))))
```

No Cheating!

```
(defthm event-loop-dart-gamep
  (implies (dart-gamep w)
           (dart-gamep (event-loop w es))))

(defthm big-bang-dart-gamep
  (dart-gamep (event-loop 3 es)))
```

No Cheating!

```
(defthm big-bang-dart-gamep
  (dart-gamep (event-loop 3 es))
  :hints
  (("Goal"
    :in-theory (disable event-loop-dart-gamep)
    :use (:instance event-loop-dart-gamep (w 3)))))

(defthm big-bang-darts-left
  (implies (>= (count-clicks es) 3)
           (win-or-lose (event-loop 3 es)))
  :hints
  (("Goal"
    :in-theory (disable event-loop-darts-left)
    :use (:instance event-loop-darts-left (w 3)))))
```

Extending Big Bang

```
(big-bang *WORLD_0*

  (on-draw RENDER *WIDTH* *HEIGHT*)
  (on-tick TOCK *RATE*)
  (on-key REACT)
  (on-mouse CLICK)
  (stop-when DONE)

  (world-invariant GOOD)
  (world-measure MEASURE PROGRESS))
```

Extending Big Bang

```
(world-invariant GOOD) ; becomes:

(defthm event-loop-GOOD
  (implies (GOOD w)
           (GOOD (event-loop w es))))

(defthm big-bang-GOOD
  (GOOD (event-loop *WORLD_0* es))
  :hints
  (("Goal"
    :in-theory (disable event-loop-GOOD)
    :use (:instance event-loop-GOOD
                    (w *WORLD_0*)))))
```

Extending Big Bang

```
(world-measure MEASURE PROGRESS) ; becomes:

(defun count-PROGRESS (es) ...)

(defthm event-loop-MEASURE ...)

(defthm big-bang-MEASURE ...)
```

Extending Big Bang

```
(world-measure MEASURE PROGRESS) ; becomes:

(defun count-PROGRESS (es)
  (cond ((endp es) 0)
        ((PROGRESS (car es))
         (1+ (count-PROGRESS (cdr es))))
        (t (count-PROGRESS (cdr es)))))

(defthm event-loop-MEASURE ...)

(defthm big-bang-MEASURE ...)
```

Extending Big Bang

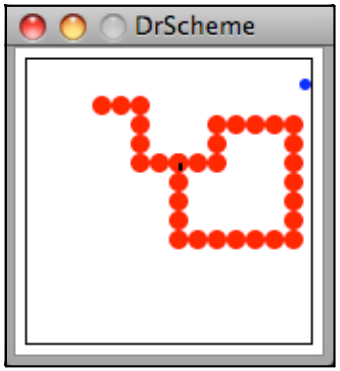
```
(world-measure MEASURE PROGRESS) ; becomes:

(defun count-PROGRESS (es) ...)

(defthm event-loop-MEASURE
  (implies (and (GOOD w)
                (>= (count-PROGRESS es)
                    (MEASURE w)))
           (DONE (event-loop w es))))

(defthm big-bang-MEASURE ...)
```

Extending Big Bang

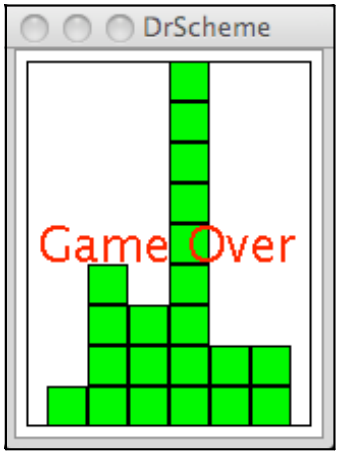
```
(world-measure MEASURE PROGRESS) ; becomes:

(defun count-PROGRESS (es) ...)

(defthm event-loop-MEASURE ...)

(defthm big-bang-MEASURE
  (implies (>= (count-PROGRESS es)
               (MEASURE *WORLD_0*))
           (DONE (event-loop *WORLD_0* es)))
  :hints
  (("Goal"
    :in-theory (disable event-loop-MEASURE)
    :use (:instance event-loop-MEASURE
                    (w *WORLD_0*)))))
```

Experiments

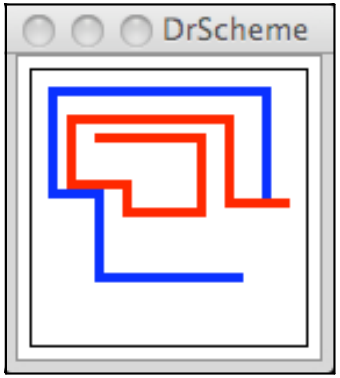
Experiments



Worm:
- all segments are adjacent
- all segments are on-screen
- no segments overlap

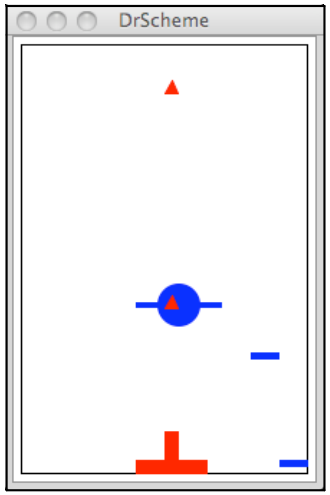


Blocks:
- no blocks overlap
- all blocks are on-screen



Bikes:
- trails run in cardinal directions

Experiments



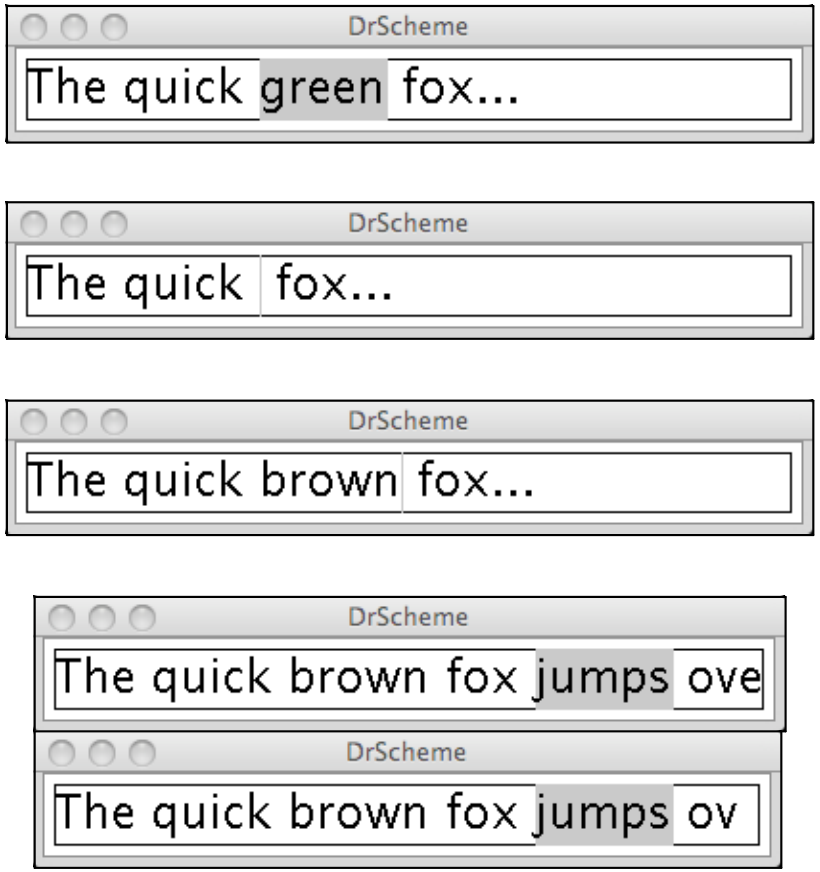
UFO:

- **all objects stay on-screen.**
- **UFO's descent acts as a time limit.**

Hangman:

- **limit of 5 (failed) + word-length (successful) keystrokes**

Experiments



Editor:

- **partial correctness of typing, selecting, deleting, and navigating**
- **no partial letters are displayed**
- **displayed text is maximal prefix**

Experiments

Project	Lines	Lemmas	CPU seconds
Hangman	365	11	1.48
Blocks	450	16	0.86
UFO	696	23	13.97
Worm	824	34	4.90
Editor	1,117	59	5.04
Bikes	1,354	84	202.11

Thank You.

Images

```
; Basic shape constructors:
(circle radius mode color)
(rectangle width height mode color)
(triangle size mode color)
(star inner outer points mode color)

; Combining shapes:
(add-line image x1 y1 x2 y2 color)
(empty-scene width height)
(place-image image x y scene)

; Predicates and accessors:
(image? x)
(image-width image)
(image-height image)
```

Images

```
(defthm circle/image?
  (implies (and (natp r)
                (mode? m)
                (image-color? c))
           (image? (circle r m c))))

(defthm circle/image-width
  (equal (image-width
          (circle radius mode color))
         (* radius 2)))

(defthm circle/image-height
  (equal (image-height
          (circle radius mode color))
         (* radius 2)))
```

Images

```
(defthm empty-text-image-width
  (implies (and (font-size? size) (image-color? color))
           (= (image-width (text "" size color)) 0)))

(defthm append-right-text-image-width
  (implies (and (stringp a) (stringp b)
                (font-size? size) (image-color? color))
           (>= (image-width (text (string-append a b) size color))
               (image-width (text a size color)))))

(defthm append-left-text-image-width
  (implies (and (stringp a) (stringp b)
                (font-size? size) (image-color? color))
           (>= (image-width (text (string-append a b) size color))
               (image-width (text b size color)))))
```