... and Patched!

Reduceron Characteristics

 The size of compiled bodies is bounded so that by using wide parallel memories bodies are instantiated in a single clock cycle.

 Primitive redexes in instances of function bodies are detected dynamically for primitive redex speculation.

Reduceron Characteristics

 The size of compiled bodies is bounded so that by using wide parallel memories bodies are instantiated in a single clock cycle.

 Primitive redexes in instances of function bodies are detected dynamically for primitive redex speculation.

 The sizes of bodies containing primitive applications are reckoned as if every primitive-redex test fails.

Detecting Guaranteed PRS Candidates Statically

Goal

Find the primitive applications whose every run-time instance is guaranteed to be a redex.

Method?

Suppose we propagate integer-value information:

- inwards from program input;
- outwards from numeric literals;
- onwards through primitive redex speculation.

Example

In `safe` we find just one guaranteed primitive redex

```
safe x d (q:qs)  =  x /= q && x /= q+d && x /= q-d &&
                    safe x (d+1) qs
```

as both `x` and `q` are drawn from data structures.

Valuable data structures

Definition

Let D be a data expression that evaluates to the construction $C\ e_1 \ldots e_n$. D is valuable if each integer component e_i is a value, and each data component e_i is valuable.

Revisiting `safe`

With information about valuable data structures, the guaranteed primitive redexes become:

```
safe x d (q:qs)  =  x /= q && x /= q+d && x /= q-d &&
                    safe x (d+1) qs
```

Non-uniformity, Cloning & Specialization

- Problem: for some applications of a function there is scope for primitive-redex speculation in the body but not for others.
- Solution: clone by need, specialising functions for different combinations of value/valuable argument positions.
- In principle, the number of clones could be exponential in the arity of a function. In practice, there is often just one specialization needed — and the original is discarded.

Spineless Queens Safety!

```
safe    v0 v1 v2              =  v2 [safe#1,safe#2] v0 v1
safe#1  v0 v1 v2 v3 v4        =
      {(/=) v3 v0}
            [safe2#5,safe2#6]
                  v3
                        {(+) v0 v4}
                              {(-) v0 v4}
                                    {(+) v4 1}
                                          v1
safe#2  v0 v1 v2              =  True
safe2#1 v0 v1 v2 v3           =  False
safe2#2 v0 v1 v2 v3           =  v3 [safe#1,safe#2] v1 v2
safe2#3 v0 v1 v2 v3 v4        =  False
safe2#4 v0 v1 v2 v3 v4        =  {(/=) v1 v2} [safe2#1,safe2#2] ...
safe2#5 v0 v1 v2 v3 v4 v5     =  False
safe2#6 v0 v1 v2 v3 v4 v5     =  {(/=) v1 v2} [safe2#3,safe2#4] ...
```

Fibonacci Presto!

```
fib   n  =
      {n<=1}
            [fib#1, fib#2]
                  n
fib#1 n  =
      {n<=3}
            [fib#1, fib#2]
                  {n-2}
                        (+)
                              ({n<=2} [fib#1, fib#2] {n-1})
fib#2 n  =
      1
```

Latest Performance Figures

Run-times in clock cycles

	ICFP paper	**Presto**
Fib	13462684	6731343
Queens	18796348	8544006

Hand-reductions per cycle

	ICFP paper	**Presto**
Fib	0.90	1.80
Queens	0.72	1.58