Ivo functions forward and backward

Nathaniel Nystrom
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
nate.nystrom@usi.ch

Igor Moreno Santos
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
igor.moreno@usi.ch

ACM Reference format:
Nathaniel Nystrom and Igor Moreno Santos. 2016. Ivo functions forward and backward. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 1 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

Data intensive applications present new challenges for programming languages. These applications use functional programming language features (notably map, reduce, and group-by operations) for organizing computation across multiple nodes. In addition, they require features for searching, filtering, and querying data.

Ivo is a new programming language designed for these applications. Ivo supports flexible syntax for defining new operators and control-flow constructs. Pattern matching and logic programming features allow for declarative data querying and manipulation, both sequentially and in parallel. The key constructs that enable these features are *mixfix functions* and *backward functions*.

Mixfix functions

Ivo function declarations define *mixfix operators*. For example, the following functions declare some boolean operators:

```
def if (Boolean) then {a} else {{a}} -> a
def ((Boolean)) || {Boolean} -> Boolean
def ((Boolean)) && {Boolean} -> Boolean
def ! (Boolean) -> Boolean
```

Syntax is defined without an explicit grammar: the order of declaration specifies the precedence: in this case `(if _ then _ else _)` has lowest precedence and `(! _)` has the highest precedence. Double parentheses around a parameter indicate associativity. Curly brackets in the declarations specify that the parameter is call-by-name rather than call-by-value, allowing declaration of short-circuiting operators and other control structures.

Backward functions

The second enabling feature of Ivo is *backward functions*. A traditional function in Ivo is referred to as a forward function, taking inputs and returning an output. A backward function takes an output and returns one or more inputs. A backward function can be invoked by passing in its return value and optionally some arguments, yielding the other arguments. Backward functions can therefore be used in patterns and in logical formulas. This language feature is based on a similar feature in the language JMatch (Liu and Myers 2003), but simplified and extended.

As an example, we can write the following code to compute the *n*th power of a number in logarithmic time:

```
def (Int) ** ((Nat)) -> Int
    x ** 0       = 1
    x ** (2*k)   = square (x ** n)
    x ** (2*k+1) = x * x ** (2*k)
```

When invoked as *x* `**` *n*, the `2*k` pattern invokes the backward `(_ * _)` function, passing in 2 and the argument *n* and returning a value to bind to the *unknown* variable `k`. The pattern will match only if *n* is even. Similarly, the pattern `2*k+1` will match only if *n* is odd.

A backward mode can also generate more than one result, allowing construction and iteration over of *streams*. A `for` expression takes a *formula*—a boolean pattern with one or more unknowns—and a body expression. The body of the `for` is evaluated for each satisfying assignment of the formula. For example,

```
for (x in xs && x > 0) print x
```

iterates through and prints each element of the collection `xs` that is greater than 0. The `in` formula is actually just an invocation of a backward function that returns a stream of all elements in a collection. The `&&` operator in the formula filters the stream of bindings to `x`.

The `for` construct iterates through all solutions of a formula, evaluating the body for each solution. Ivo supports parallel execution using the `for async` construct, which evaluates body for each solution, in parallel. Concurrency in general is supported using the finish/async model of X10 (Charles et al. 2005).

References

Philippe Charles, Christian Grothoff, Christopher Donawa, Kemal Ebcioğlu, Allan Kielstra, Christoph von Praun, Vijay Saraswat, and Vivek Sarkar. 2005. X10: An object-oriented approach to non-uniform cluster computing. In *Proceedings of the 2005 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA).*

Jed Liu and Andrew C. Myers. 2003. JMatch: Abstract Iterable Pattern Matching for Java. In *Proc. 5th Int'l Symp. on Practical Aspects of Declarative Languages (PADL).* New Orleans, LA, 110–127.

Conference'17, Washington, DC, USA
2016. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn