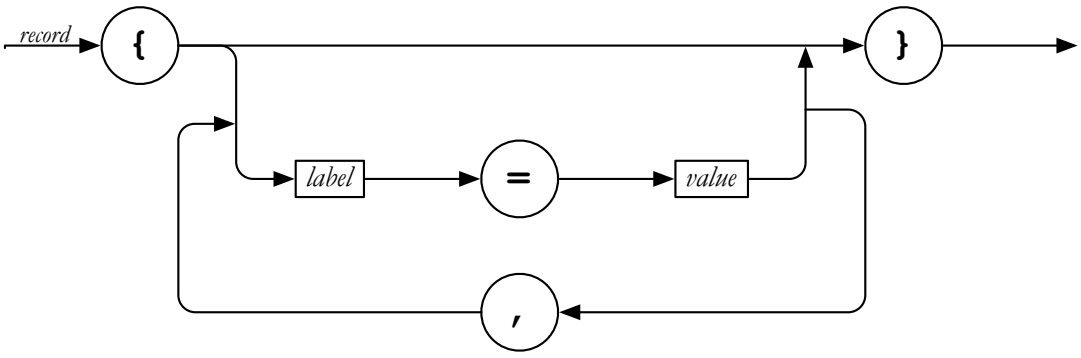

Figure 1: Syntax diagram for SML record values

More Features of Standard ML — Records
Fall 2013

1 Introduction

These notes are being created in conjunction with the teaching of CPSC 370 in the Fall 2013 term at the University of Northern British Columbia.

These notes are a work in progress, and copyright belongs exclusively to David Casperson.

2 Records in Standard ML

Records are the nearest STANDARD ML equivalent of C++ `structs` or PASCAL `records`. Like other ML constructs, records give rise to values, patterns, and types. Record value expressions look like a set of equations.

2.1 Standard ML record values

```
{name="Fred", age=34, female=false} ;
{age=34, female=false, name="Fred"} ;
```

Unlike `structs` in C++ and records in Pascal, the value of a record does not depend on the order of the label expressions. (Both of the above

values are the same). The formal syntax of records is given in Figure 1. The syntax of a label is *either* a legal Standard ML identifer, *or* an unsigned positive integer.

The kinds of things that one can do with records are about the same as with `structs` in C++ or records in Pascal; that is to say if you treat them as black boxes, you can store them in variables; put them inside other structures. In order to do something other than treat them as black boxes, you need to be able to take them apart.

Like datatypes, the way to take them apart is with patterns, which look a lot the corresponding values.

2.2 Standard ML record patterns

For the record

```
{name="Fred", age=34, female=false} ;
```

the corresponding pattern is

{`name=`p_1`, age=`p_2`, female=`p_3}

where p_1, p_2, and p_3 are appropriate sub-patterns. For instance, we could write

```
fun isMale {name=_, age=_, female=f} = not f ;
```

Again, order of the equations does not matter.

{`age=`p_2`, name=`p_1`, female=`p_3}

matches the same as the above pattern.

In patterns for records, there a number of flavours of syntactic sugar. When label is an identifier the pattern fragment `label` is short for `label=label`. For instance,

```
fun isMale {name=_, age=_, female} = not female ;
```

also works.

Where the type is known, labels may be omitted entirely and replaced by "`...`". For instance, the following two functions are equivalent.

```
fun woman {female=false,...} = false
  | woman {age=n, female=_, name=_} = (n >= 18)

fun woman {age,female,name=_} = female andalso age >= 18 ;
```

However

```
fun woman {female, age, ...} = female andalso age >= 18 ;
```

will not compile because the compiler cannot deduce how many records are missing, or their labels.

2.3 Standard ML record types

Record values have corresponding types, which have similar syntax. For intance, the type of the `woman` function above is

```
{age:int, female:bool, name:string} -> bool
```

2.3.1 Tuples are syntactic sugar for records

A tuple is just a record whose labels form a consecutive sequence of integers 1 to n. So, for instance, the tuple

```
(34,false,"Fred")
```

is syntactic sugar for

```
{1=34,2=false,3="Fred"}
```

(and "`()`" is syntactic sugar for "`{}`").

2.3.2 Extractor notation

In *expressions*, `#label` is syntactic sugar for the function `(fn {label=xx,...}=>xx)`. For instance, after

```
  val fred = {name="Fred", age=34, female=false} ;
```

`#name fred` returns `"Fred"` and `#age fred` returns `34`. The same caveat for "`...`" as above applies; you can only use this syntax in situations where you know the full record type. This frequently happens when you combine datatypes and records. Suppose for instance that we define binary trees by

```
datatype 'a tree = Empty
                 | Node of {left='a tree, value='a, right='a tree};
```

We can then write a function to extract a tree's values in pre-order as follows:

```
fun preOrder Empty = []
  | preOrder (Node tree)
    = #value tree :: preOrder (#left tree) @ preOrder (#right tree) ;
```

Note that the above works, because the variable `tree` has a known record type.

Also note how the code is much easier to read with records.

The field extractor notation can also be used with tuples, so `#2 (3,"cat",false)` is `"cat"`.