```
            else collect_duplicates A_ xs (z :: ys) zs)
      else collect_duplicates A_ (z :: xs) (z :: ys) zs);

end; (*struct Example*)
```

Obviously, polymorphic equality is implemented the Haskell way using a type class. How is this achieved? HOL introduces an explicit class *equal* with a corresponding operation *equal-class.equal* such that *equal-class.equal* = (=). The preprocessing framework does the rest by propagating the *equal* constraints through all dependent code equations. For datatypes, instances of *equal* are implicitly derived when possible. For other types, you may instantiate *equal* manually like any other type class.

2.5 Explicit partiality

Partiality usually enters the game by partial patterns, as in the following example, again for amortised queues:

definition *strict-dequeue* :: *'a queue* ⇒ *'a* × *'a queue* **where**
  strict-dequeue q = (*case dequeue q*
    of (*Some x*, *q'*) ⇒ (*x*, *q'*))

lemma *strict-dequeue-AQueue* [*code*]:
  strict-dequeue (*AQueue xs* (*y* # *ys*)) = (*y*, *AQueue xs ys*)
  strict-dequeue (*AQueue xs* []) =
    (*case rev xs of y* # *ys* ⇒ (*y*, *AQueue* [] *ys*))
  by (*simp-all add*: *strict-dequeue-def*) (*cases xs*, *simp-all split*: *list.split*)

In the corresponding code, there is no equation for the pattern *AQueue* [] []:

```
strict_dequeue :: forall a. Queue a -> (a, Queue a);
strict_dequeue (AQueue xs []) = (case reverse xs of {
                                  y : ys -> (y, AQueue [] ys);
                                });
strict_dequeue (AQueue xs (y : ys)) = (y, AQueue xs ys);
```

In some cases it is desirable to state this pseudo-"partiality" more explicitly, e.g. as follows:

axiomatization *empty-queue* :: *'a*

definition *strict-dequeue'* :: *'a queue* ⇒ *'a* × *'a queue* **where**

strict-dequeue' q = (*case dequeue q of* (*Some x*, *q'*) ⇒ (*x*, *q'*)
| - ⇒ *empty-queue*)

lemma *strict-dequeue'-AQueue* [*code*]:
strict-dequeue' (*AQueue xs* []) = (*if xs* = [] *then empty-queue*
else strict-dequeue' (*AQueue* [] (*rev xs*)))
strict-dequeue' (*AQueue xs* (*y* # *ys*)) =
(*y*, *AQueue xs ys*)
by (*simp-all add*: *strict-dequeue'-def split*: *list.splits*)

Observe that on the right hand side of the definition of *strict-dequeue'*, the unspecified constant *empty-queue* occurs. An attempt to generate code for *strict-dequeue'* would make the code generator complain that *empty-queue* has no associated code equations. In most situations unimplemented constants indeed indicated a broken program; however such constants can also be thought of as function definitions which always fail, since there is never a successful pattern match on the left hand side. In order to categorise a constant into that category explicitly, use the *code* attribute with *abort*:

declare [[*code abort*: *empty-queue*]]

Then the code generator will just insert an error or exception at the appropriate position:

```
empty_queue :: forall a. a;
empty_queue = error "Foundations.empty_queue";

strict_dequeue :: forall a. Queue a -> (a, Queue a);
strict_dequeue (AQueue xs (y : ys)) = (y, AQueue xs ys);
strict_dequeue (AQueue xs []) =
  (if null xs then empty_queue
    else strict_dequeue (AQueue [] (reverse xs)));
```

This feature however is rarely needed in practice. Note that the HOL default setup already includes

declare [[*code abort*: *undefined*]]

– hence *undefined* can always be used in such situations.