More Event Combinators

CML provides two more event combinators: `guard` and `withNack`:

```
val guard    : (unit -> 'a event)       -> 'a event
val withNack : (unit event -> 'a event) -> 'a event
```

Each time an event involving `guard` f is synchronized on, `guard` f is replaced by $f\,()$, which is then processed further.

`guard` may be used, e.g., to generate a fresh reply channel each time an event is synchronized on.

More Event Combinators (Cont.)

```
val guard    : (unit -> 'a event)       -> 'a event
val withNack : (unit event -> 'a event) -> 'a event
```

Each time an event involving `withNack` f is synchronized on, `withNack` f is replaced by $f\ nev$, which is then processed further, where nev is a fresh *negative acknowledgment* `unit event` that will become enabled if some event other than $f\ nev$ is selected in the synchronization.

For example, if `choose`$[ev,$ `withNack` $f]$ is synchronized on, but ev is eventually selected, then the negative acknowledgment event nev passed to f will become enabled, indicating that any operation begun by $f\ nev$ should be aborted.

More Event Combinators (Cont.)

```
val guard    : (unit -> 'a event)        -> 'a event
val withNack : (unit event -> 'a event) -> 'a event
```

Each time an event involving `withNack` f is synchronized on, `withNack` f is replaced by $f\ nev$, which is then processed further, where nev is a fresh *negative acknowledgment* `unit event` that will become enabled if some event other than $f\ nev$ is selected in the synchronization.

For example, if `choose`$[ev,$ `withNack` $f]$ is synchronized on, but ev is eventually selected, then the negative acknowledgment event nev passed to f will become enabled, indicating that any operation begun by $f\ nev$ should be aborted.

The guard functions of `guard`'s and `withNack`'s should run quickly; in particular, they shouldn't block.