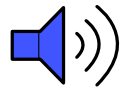
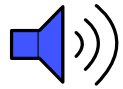
Solution to STM assignment



- The finish function passes an IO () action `canceller` to a list of request actions and run them concurrently.
- The canceller is implemented with an STM that allows each request to report its progress to the finish thread, which makes decision on request cancellation.

```
1  finish :: [IO () -> IO b] -> Int -> IO [b]
2  main = do
3    let urls = [("http://www.yahoo.com/", "test1.txt"),
4                ("http://www.bing.com/", "test4.txt")]
5    let actions = map request urls
6    ms <- finish actions 1
7    mapM_ putStrLn ms
8   where
9      request :: (String, String) -> IO () -> IO String
10     request (url, f) canceller = do
11           d <- simpleHttp url
12           canceller
13           L.writeFile f d
14           return $ url ++ " done"
15           `catch` handler
16           where handler = \(SomeException e) ->
17              return (url ++ ": " ++ displayException e)
```

TVar for communication 

- Use a single TVar ‘committed’ for communication between requests and finish thread.

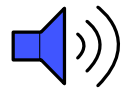
```
1  finish :: [IO () -> IO b] -> Int -> IO [b]
2  finish actions n =
3        -- the TVar contains a list of IDs of the requests
4        -- that have committed (i.e. completed download).
5    do committed <- newTVarIO []
6
7        -- helper action to read the IDs
8        let get = readTVar committed
9
10       -- helper action to find the length of the ID list
11       let len = length <$> get
12
13       -- helper action to add an ID 'x' to the ID list
14       let add x = modifyTVar' committed (x :)
15
16       -- more code
```

TVar for communication

- Run STM ‘atomically’ and block with ‘retry’.
- request STM blocks if more than n requests have committed.
- finish STM blocks if less than n requests have committed.

```
1  finish :: [IO () -> IO b] -> Int -> IO [b]
2  finish actions n =
3    do committed <- newTVarIO []
4       let get = readTVar committed
5       let len = length <$> get
6       let add x = modifyTVar' committed (x :)
7
8       -- add ID 'i' and block if n requests have committed
9       let commit i = do { add i; l <- len; when (l > n) retry }
10
11      as <- mapM (\(f, i) -> async $ f $ atomically $ commit i)
12          $ zip actions [1..]
13
14      lst <- atomically $ do l <- len
15                             if l < n -- less than n committed
16                                -- block with retry
17                                then retry
18                                -- return ID list
19                                else get
20      -- more code
```

TVar for communication



- Once n requests have commmited, the rest is cancelled.

```
1  finish :: [IO () -> IO b] -> Int -> IO [b]
2  finish actions n =
3    do committed <- newTVarIO []
4       let get = readTVar committed
5       let len = length <$> get
6       let add x = modifyTVar' committed (x :)
7       let commit i = do { add i; l <- len; when (l > n) retry }
8
9       as <- mapM (\(f, i) -> async $ f $ atomically $ commit i)
10          $ zip actions [1..]
11
12      lst <- atomically
13          $ do { l <- len; if l < n then retry else get }
14
15      -- cancel requests whose IDs are not in the list
16      mapM_ (\(a, i) -> unless (i `elem` lst) $ cancel a)
17          $ zip as [1..]
18
19      -- wait for all request threads to complete
20      mapM wait as
```