RAC3, A Real World Use Case

aka

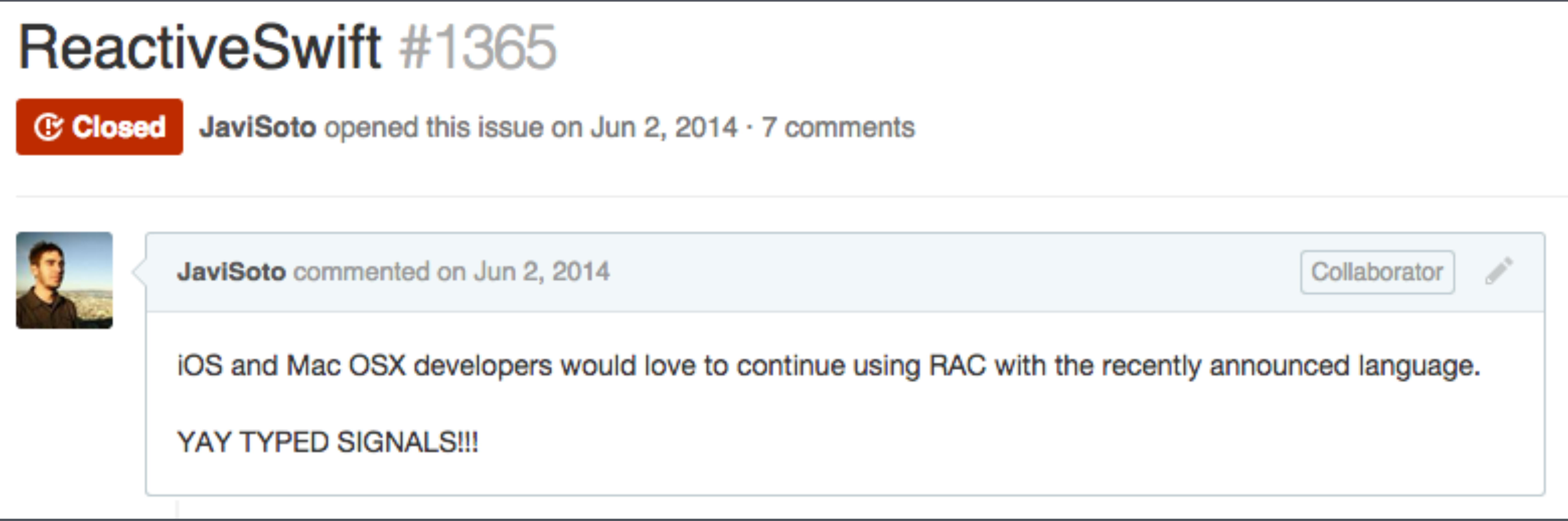
ReactiveChess

Javier Soto (@Javi)

iOS at Twitter

Prediction:

All of the code in these slides won't compile on the latest Xcode by the time you're reading this.



ReactiveSwift #1365

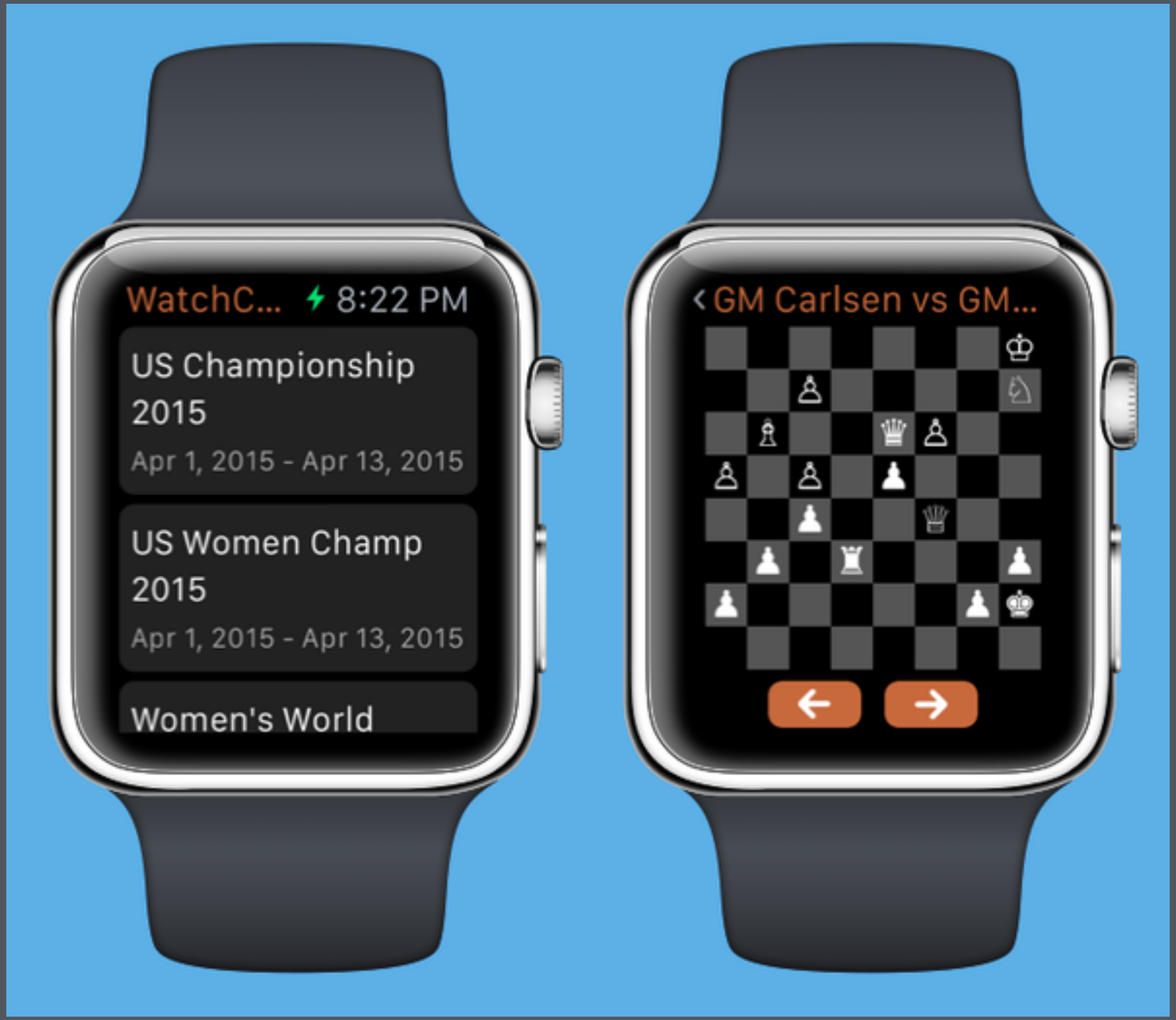
Closed JaviSoto opened this issue on Jun 2, 2014 · 7 comments

JaviSoto commented on Jun 2, 2014 Collaborator

iOS and Mac OSX developers would love to continue using RAC with the recently announced language.

YAY TYPED SIGNALS!!!

WatchChess.app



WatchChess.app

- ReactiveCocoa 3.0
- Argo

History

- First prototype: Swift 1.2 with Xcode 6.3 Beta 1
- v1.0: Swift 1.1, RAC 3.0 pre-alpha
- v1.1: Swift 1.2, RAC 3.0-beta.1
- v1.2: RAC 3.0-beta.6

Big Takeaways

- Typed Signals
- Less debugging required
- Conciseness
- Clearer semantics

Conciseness

```
[[[[client
    logInUser]
    flattenMap:^(User *user) {
        return [client loadCachedMessages:user];
    }]
    flattenMap:^(NSArray *messages) {
        return [client fetchMessagesAfter:messages.lastObject];
    }]
    subscribeNext:^(NSArray *newMessages) {
        NSLog(@"New messages: %@", newMessages);
    } completed:^{
        NSLog(@"Fetched all messages.");
    }];
```

Conciseness

```
client.loginUser()
    |> flatMap(.Latest) { client.loadCachedMessages($0) }
    |> flatMap(.Concat) { client.fetchMessagesAfter($0.last) }
    |> start(next: { println("New messages: \($0)") },
         completed: { println("Fetched all messages.") })
```

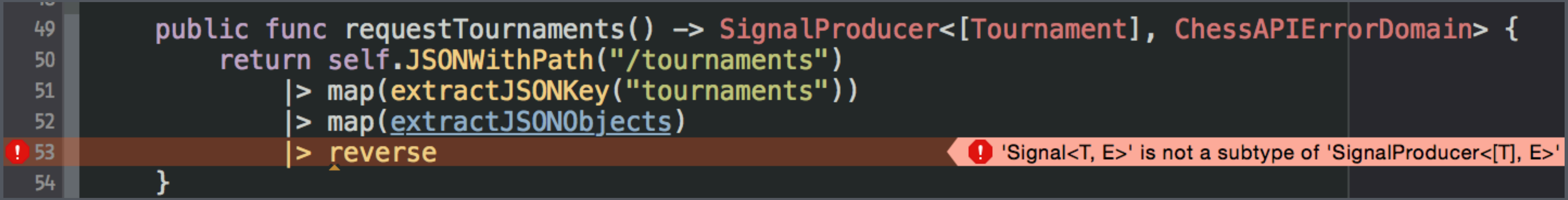
Biggest sources of frustration

- Compiler crashes
- Type errors

Figure out type errors in RAC 3 with this one weird trick

Type Errors

WAT

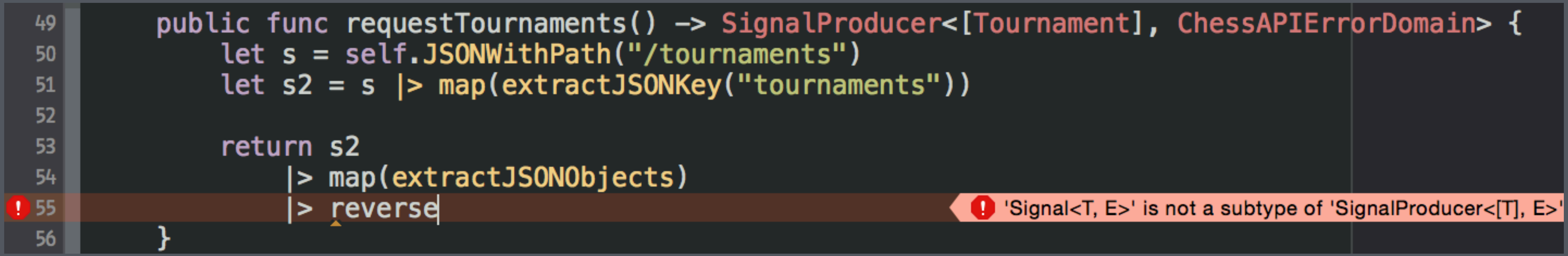


```
    public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
        return self.JSONWithPath("/tournaments")
            |> map(extractJSONKey("tournaments"))
            |> map(extractJSONObjects)
            |> reverse
    }
```

'Signal<T, E>' is not a subtype of 'SignalProducer<[T], E>'

Type Errors

Extract intermediate results into separate values



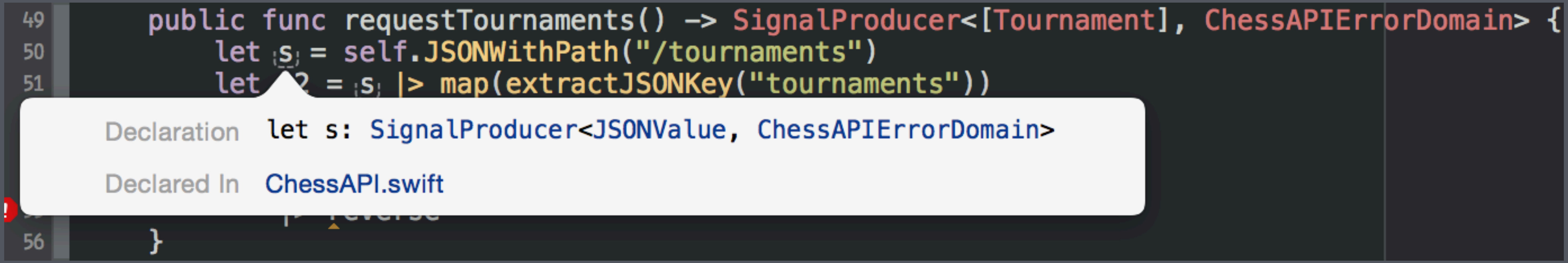
```
    public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
        let s = self.JSONWithPath("/tournaments")
        let s2 = s |> map(extractJSONKey("tournaments"))

        return s2
            |> map(extractJSONObjects)
            |> reverse
    }
```

'Signal<T, E>' is not a subtype of 'SignalProducer<[T], E>'

Type Errors

Inspect the types (⌥ + click)



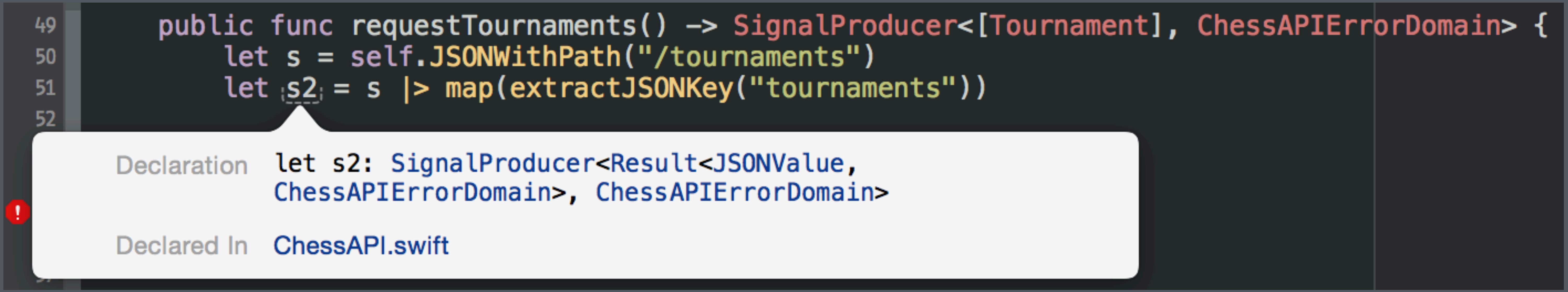
```
    public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
        let s = self.JSONWithPath("/tournaments")
        let s2 = s |> map(extractJSONKey("tournaments"))
            |> reverse
    }
```

Declaration let s: SignalProducer<JSONValue, ChessAPIErrorDomain>

Declared In ChessAPI.swift

Type Errors

Inspect the types (⌥ + click)



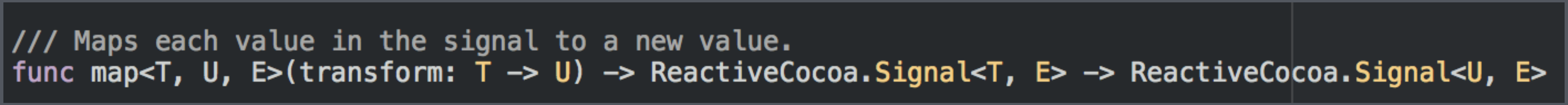
```
    public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
        let s = self.JSONWithPath("/tournaments")
        let s2 = s |> map(extractJSONKey("tournaments"))
```

Declaration `let s2: SignalProducer<Result<JSONValue, ChessAPIErrorDomain>, ChessAPIErrorDomain>`

Declared In ChessAPI.swift

Type Errors

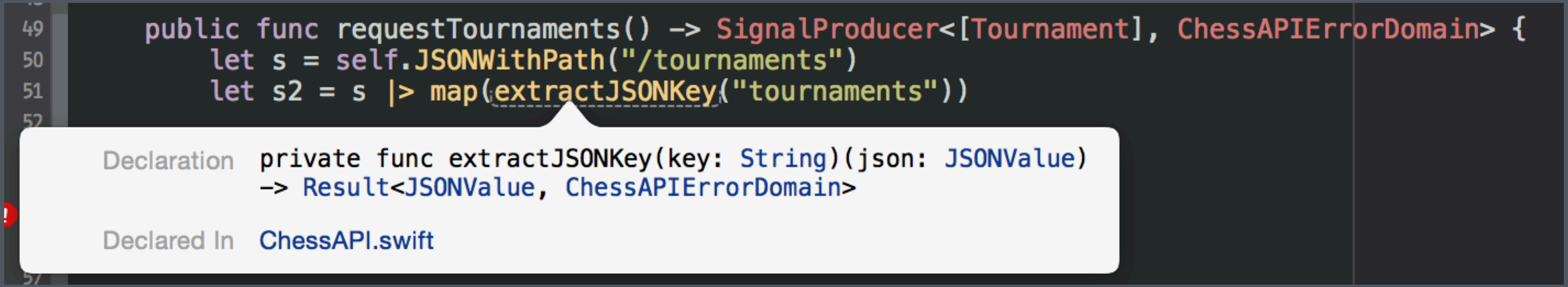
Check RAC's function signatures (⌘ + click)



```
/// Maps each value in the signal to a new value.
func map<T, U, E>(transform: T -> U) -> ReactiveCocoa.Signal<T, E> -> ReactiveCocoa.Signal<U, E>
```

Type Errors

Check if your types match those expected by RAC



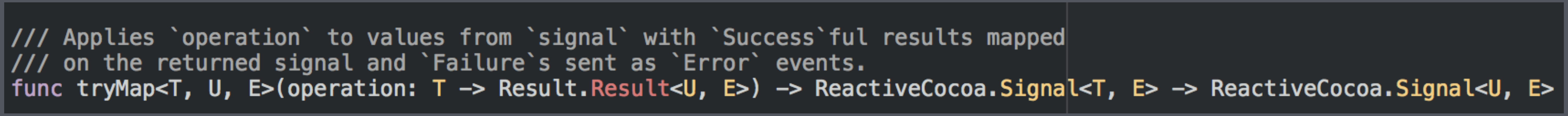
```
    public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
        let s = self.JSONWithPath("/tournaments")
        let s2 = s |> map(extractJSONKey("tournaments"))
```

Declaration
```
private func extractJSONKey(key: String)(json: JSONValue)
-> Result<JSONValue, ChessAPIErrorDomain>
```

Declared In ChessAPI.swift

Type Errors

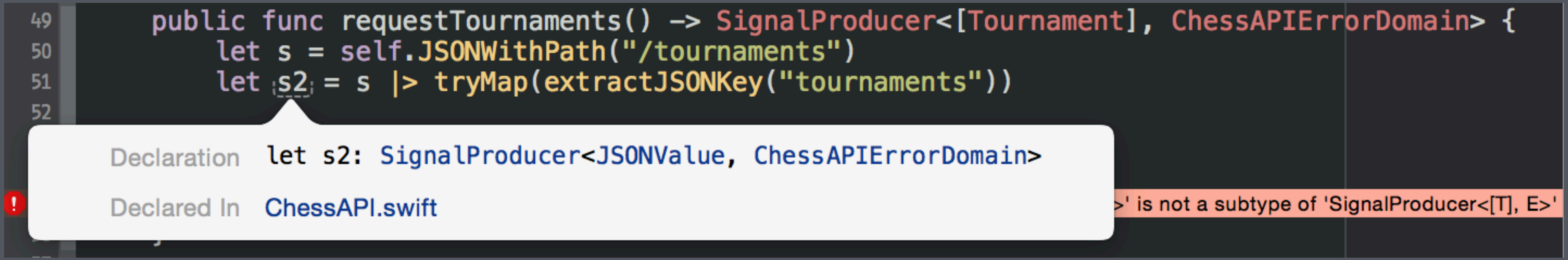
Look for a function that matches what you're trying to do



```
/// Applies `operation` to values from `signal` with `Success`ful results mapped
/// on the returned signal and `Failure`s sent as `Error` events.
func tryMap<T, U, E>(operation: T -> Result.Result<U, E>) -> ReactiveCocoa.Signal<T, E> -> ReactiveCocoa.Signal<U, E>
```

Type Errors

If it compiles, it works.



```
public func requestTournaments() -> SignalProducer<[Tournament], ChessAPIErrorDomain> {
    let s = self.JSONWithPath("/tournaments")
    let s2 = s |> tryMap(extractJSONKey("tournaments"))
```

Declaration `let s2: SignalProducer<JSONValue, ChessAPIErrorDomain>`

Declared In ChessAPI.swift

```
>' is not a subtype of 'SignalProducer<[T], E>'
```

ChessAPI.swift

ChessAPI.swift

```swift
enum APIError: ErrorType {
  case NetworkError(NSError)
  case JSONParsingError(NSError)
  case InvalidJSONStructure(reason: String)
}
```

ChessAPI.swift

```swift
import ReactiveCocoa
import Argo

func tournaments() -> SignalProducer<[Tournament], APIError> {
    return self.JSONWithPath("/tournaments")
        |> attemptMap(extractJSONKey("tournaments"))
        |> attemptMap(extractJSONObjects)
        |> reverse
}
```

ChessAPI.swift

```swift
func JSONWithPath(path: String) -> SignalProducer<JSON, APIError> {
    let request = APIRequestWithPath(path)
    let URL = request.URL

    return self.urlSession.rac_dataWithRequest(request)
        |> log(started: "Started: \(URL)",
               completed: "Finished: \(URL)")
        |> mapError { APIError.NetworkError($0) }
        |> map { $0.0 }
        |> attemptMap(parseJSON)
    }
```

ChessAPI.swift

```swift
func parseJSON(data: NSData) -> Result<JSON, APIError> {
    var error: NSError?
    if let json = NSJSONSerialization(data, error: &error) {
        return success(JSON.parse(json))
    }
    else {
        return failure(APIError.JSONParsingError(error!))
    }
}
```

Custom RAC Operators

Custom RAC Operators

```
return self.urlSession.rac_dataWithRequest(request)
       |> log(started: "Started: \(URL)",
              completed: "Finished: \(URL)")
```

Custom RAC Operators

`log()`:

```
func log<T, E: ErrorType>(started: String = "",
                          next: String = "",
                          completed: String = "")
                          (producer: SignalProducer<T, E>)
                          -> SignalProducer<T, E> {
    return producer
        |> on(started: { println(started) },
              next: { println(next + " \($0)") },
              completed: { println(completed) })
}
```

Custom RAC Operators

`log()`:

```
func log<T, E: ErrorType>(started: String = "",
                          next: String = "",
                          completed: String = "")
                          (producer: SignalProducer<T, E>)
                          -> SignalProducer<T, E>
```

```
(String, String, String) -> SignalProducer -> SignalProducer
```

Custom RAC Operators

`log()`:

```
(String, String, String) -> SignalProducer -> SignalProducer
log("someMessage"): SignalProducer -> SignalProducer
```

`|>`:

```
(SignalProducer, (SignalProducer -> SignalProducer)) -> SignalProducer
// Infix order would be...
// SignalProducer |> (SignalProducer -> SignalProducer) -> SignalProducer
```

Custom RAC Operators

```
return self.urlSession.rac_dataWithRequest(request)
       |> log(started: "Started: \(URL)",
              completed: "Finished: \(URL)")
```

WatchKit Controllers

```
import WatchKit
class TournamentsInterfaceController: WKInterfaceController {
  private let tournaments = MutableProperty<[Tournament]>([])

  override init() {
    super.init()

    self.tournaments.producer |> skip(1)
      |> skipRepeatedArrays
      |> start(next: { [unowned self] tournaments in
        self.updateUIWithTournaments(tournaments)
      })
  }

  override func willActivate() {
    self.tournaments <~ self.chessAPI.requestTournaments()
      |> printAndFilterErrors("Error requesting tournaments")
      |> observeOn(UIScheduler())
  }
}
```

WatchKit Controllers

```
self.tournaments <~ self.chessAPI.requestTournaments()
    // To bind to a property, the signal can't send errors.
    |> printAndFilterErrors("Error requesting tournaments")
}
```

printAndFilterErrors()

```
func printAndFilterErrors<T, E: ErrorType>(message: String)
                                          (signal: Signal<T, E>)
                                          -> Signal<T, NoError> {
    return signal
        |> on(error: { println("\(message): \($0)") })
        |> catch { SignalProducer<T, NoError>.empty }
}
```

WatchKit Controllers

```
self.tournaments.producer
    // Easy optimization to minimize Bluetooth round-trips.
    |> skipRepeatedArrays

    |> start(next: { [unowned self] tournaments in
      self.updateUIWithTournaments(tournaments)
    })
```

skipRepeatedArrays()

```
public func skipRepeatedArrays<T: Equatable, E>(signal: Signal<[T], E>)
                                       -> Signal<[T], E> {
    return signal |> skipRepeats(==)
}
```

THANKS

- Justin Spahr-Summers (@jspahrsummers)
- Nacho Soto (@NachoSoto)
- You

Questions?