Model-based тестирование протоколов и парсинг

Денис Буздалов

5 февраля 2020

Предположения докладчика

Слушатели

Предположения докладчика

Слушатели

- воспринимают много кода на слайде

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

А также, в докладе

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

А также, в докладе

- нет претензии на исчерпывающее изложение

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

А также, в докладе

- нет претензии на исчерпывающее изложение
- нет претензии на серебряную пулю

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

А также, в докладе

- нет претензии на исчерпывающее изложение
- нет претензии на серебряную пулю
- не готовые решения

Предположения докладчика

Слушатели

- воспринимают много кода на слайде
- задают вопросы сразу
- нормально воспринимают последовательное появление

А также, в докладе

- нет претензии на исчерпывающее изложение
- нет претензии на серебряную пулю
- не готовые решения
- лукавства высших порядков ;-)

Цели доклада

- Ответить на возникшие вопросы

Цели доклада

- Ответить на возникшие вопросы
- Сделать, чтобы возникли новые

Цели доклада

- Ответить на возникшие вопросы
- Сделать, чтобы возникли новые, по крайней мере попытаться

Цели доклада

- Ответить на возникшие вопросы
- Сделать, чтобы возникли новые, по крайней мере попытаться
- Выровнять понимание у всех:
  - кто пишет спецификации
  - кто реализует интерпретатор спецификаций
  - вообще, интересующимся функциональным программистам

Цели доклада

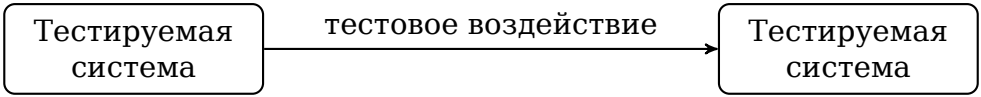
- Ответить на возникшие вопросы
- Сделать, чтобы возникли новые, по крайней мере попытаться
- Выровнять понимание у всех:
  - кто пишет спецификации
  - кто реализует интерпретатор спецификаций
  - вообще, интересующимся функциональным программистам

Каждый может узнать что-то новое, но будет слышать уже известное

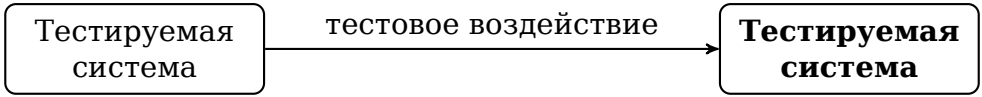
На повестке дня

- Тестовая система
  - model-based testing
  - model-based генерация тестов
  - тестирование протоколов
- Дырочность интерфейсов
- Парсинг
  - Комбинаторы парсеров
  - Дырочность
  - Двусторонние парсеры

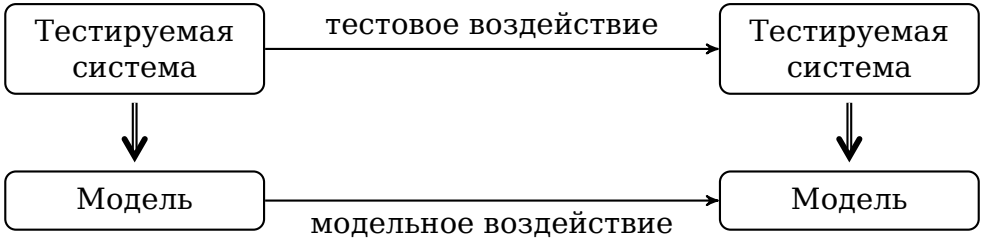
Контекст: тестирование



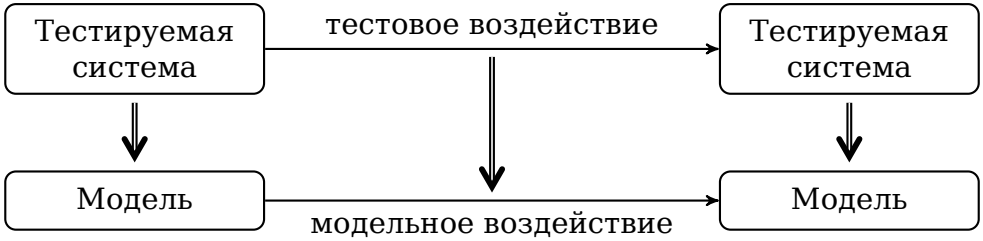
Контекст: тестирование



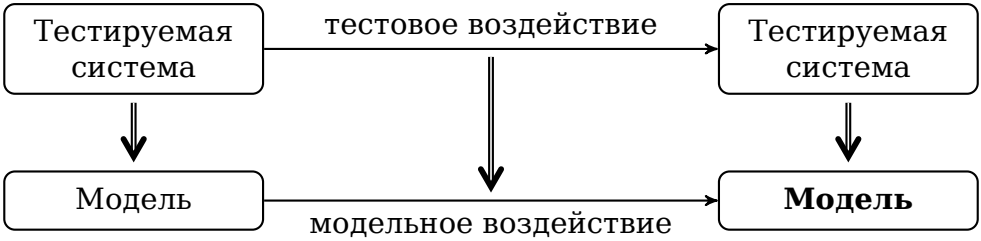
Model-based testing, ”обычный”



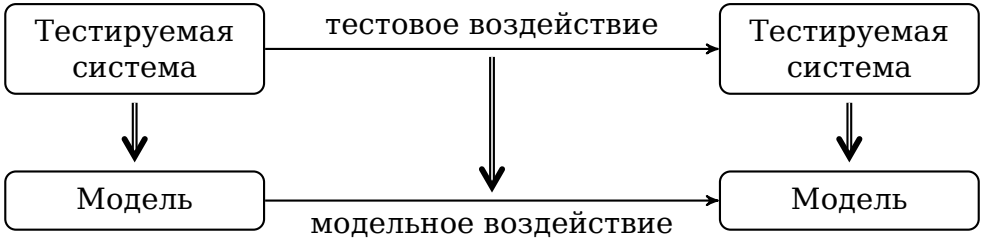
Model-based testing, ”обычный”



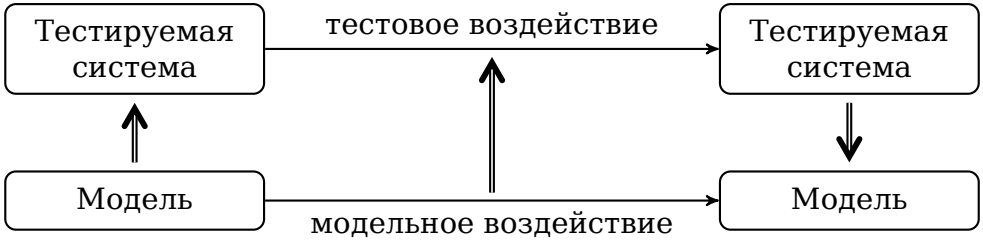
Model-based testing, ”обычный”



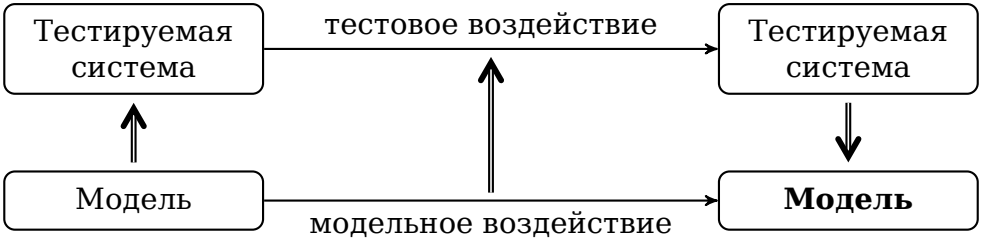
Model-based testing, ”обычный”



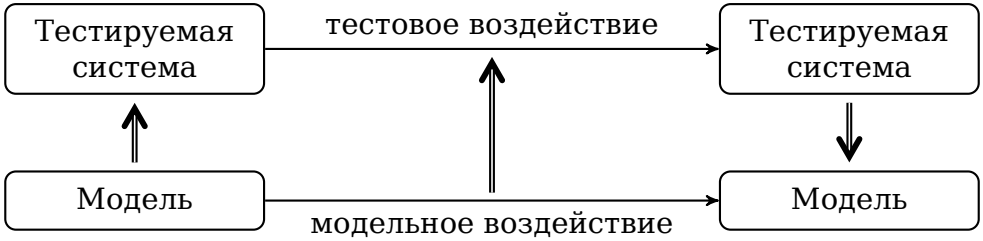
Model-based testing, стиль UniTESK



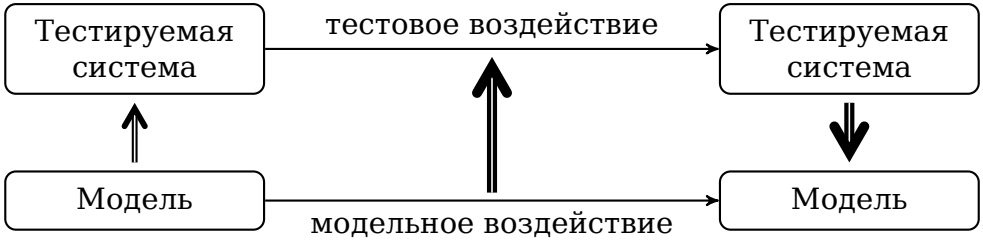
Model-based testing, стиль UniTESK



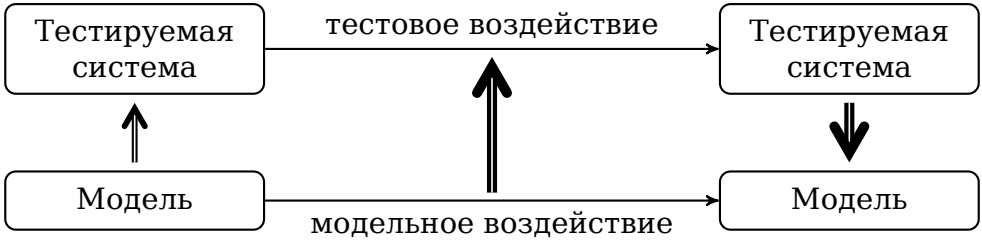
Model-based testing, стиль UniTESK

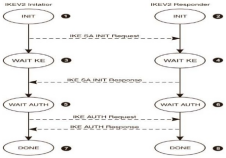


Model-based testing, стиль UniTESK

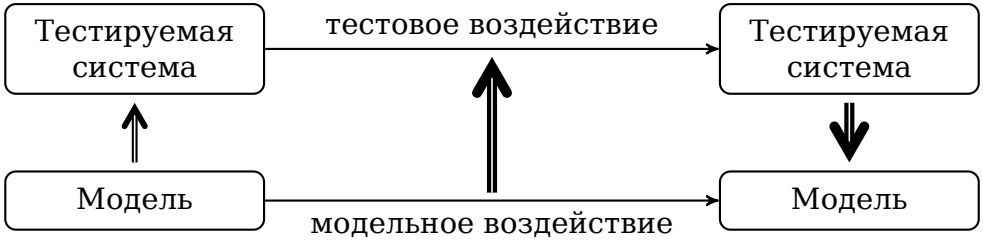


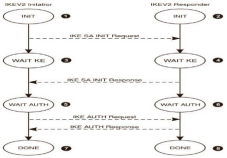
Model-based testing, стиль UniTESK





Model-based testing, стиль UniTESK



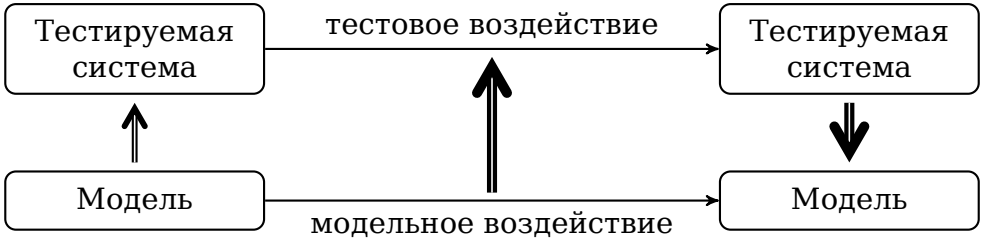


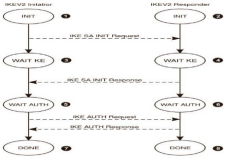
⇔

```
data SshState
  = Start
  | Versioned
  | Kexed { ...
```

⇔

Model-based testing, стиль UniTESK

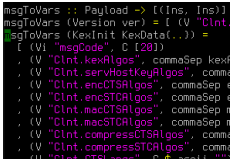




⇔

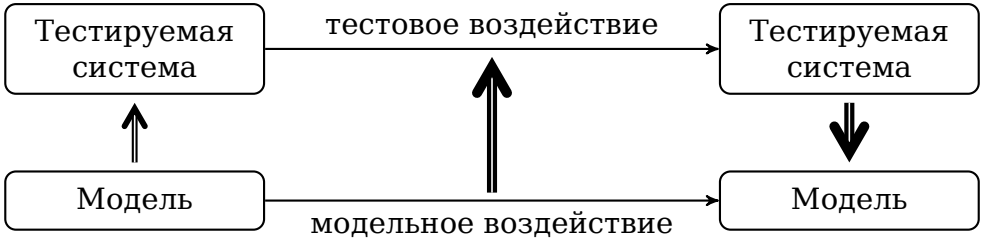
```
data SshState
  = Start
  | Versioned
  | Kexed { ...
```

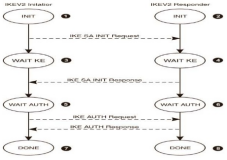
⇔



⇔

Model-based testing, стиль UniTESK

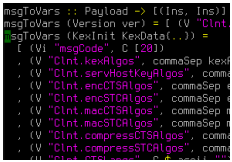




⇔

```
data SshState
  = Start
  | Versioned
  | Kexed { ...
```

⇔



```
msgToVars :: Payload -> [(Ins, Ins)]
msgToVars (Version ver) = [ (V "Clnt.
msgToVars (KexInit KexData{..}) =
  [ (Vi "msgCode", C [20])
  , (V "Clnt.kexAlgos", commaSep kexA
  , (V "Clnt.servHostKeyAlgos", comma
  , (V "Clnt.encCTSAlgos", commaSep e
  , (V "Clnt.encSTCAlgos", commaSep e
  , (V "Clnt.macCTSAlgos", commaSep m
  , (V "Clnt.macSTCAlgos", commaSep m
  , (V "Clnt.compressCTSAlgos", comma
  , (V "Clnt.compressSTCAlgos", comma
```

⇔

Сколько дырок?

Функции $a \to b$ с побочным эффектом x

Функции $a \to b$ с побочным эффектом x

```
a → (b, x)
```

Функции $a \to b$ с побочным эффектом x

```
a → (b, x)
```

```
a → x b
```

Функции $a \to b$ с побочным эффектом x

```
a → (b, x)
```

```
a → x b
```

```
x (a → b)
```

Функции $a \to b$ с побочным эффектом x

```
a → (b, x)
```

```
a → x b
```

```
x (a → b)
```

```
x a → x b
```

Функции $a \to b$ с побочным эффектом x

```
a → (b, x)
```

```
a → x b
```

```
x (a → b)
```

```
x a → x b
```

```
x a b
```

Однодырочные интерфейсы

```
f :: * → *
```

- (ковариантный) функтор

```
(a → f b) → (b → c)   → a → f c
```

Однодырочные интерфейсы

```
f :: * → *
```

- (ковариантный) функтор
  `(a → f b) → (b → c) → a → f c`
- контравариантный функтор
  `(a → f b) → (c → b) → a → f c`

Однодырочные интерфейсы

```
f :: * → *
```

- (ковариантный) функтор
  ```
  (a → f b) → (b → c)   → a → f c
  ```
- контравариантный функтор
  ```
  (a → f b) → (c → b)   → a → f c
  ```
- applicative
  ```
  (a → f b) → (a → f c) → a → f (b, c)
                          a → f a
  ```

Однодырочные интерфейсы

`f :: * → *`

- (ковариантный) функтор
  ```
  (a → f b) → (b → c)   → a → f c
  ```
- контравариантный функтор
  ```
  (a → f b) → (c → b)   → a → f c
  ```
- applicative
  ```
  (a → f b) → (a → f c) → a → f (b, c)
                          a → f a
  ```
- alternative
  ```
  (a → f b) → (a → f b) → a → f b
                          a → f b
  ```

Однодырочные интерфейсы

`f :: * → *`

- (ковариантный) функтор
  `(a → f b) → (b → c) → a → f c`
- контравариантный функтор
  `(a → f b) → (c → b) → a → f c`
- applicative
  `(a → f b) → (a → f c) → a → f (b, c)`
  `a → f a`
- alternative
  `(a → f b) → (a → f b) → a → f b`
  `a → f b`
- selective
  `f (a → c) → f (b → c) → f (Either a b) → f c`

Однодырочные интерфейсы

`f :: * → *`

- (ковариантный) функтор
  ```
  (a → f b) → (b → c)   → a → f c
  ```
- контравариантный функтор
  ```
  (a → f b) → (c → b)   → a → f c
  ```
- applicative
  ```
  (a → f b) → (a → f c) → a → f (b, c)
                          a → f a
  ```
- alternative
  ```
  (a → f b) → (a → f b) → a → f b
                          a → f b
  ```
- selective
  ```
  f (a → c) → f (b → c) → f (Either a b) → f c
  ```
- monad
  ```
  (a → f b) → (b → f c) → a → f c
  ```

Двудырочные интерфейсы

`ar :: * → * → *`

- категории

```
id    ::                        ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

Двудырочные интерфейсы `ar :: * → * → *`

- категории

```
id    ::                      ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c   → ar a (b, c)
```

Двудырочные интерфейсы `ar :: * → * → *`

- категории

```
id    ::                        ar a a
(>>>) :: ar a b → ar b c     → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c     → ar a (b, c)
(***) :: ar a b → ar c d     → ar (a, c) (b, d)
```

Двудырочные интерфейсы `ar :: * → * → *`

- категории

```
id    ::                      ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c   → ar a (b, c)
(***) :: ar a b → ar c d   → ar (a, c) (b, d)
```

- категории с копродуктами

```
(|||) :: ar a c → ar b c   → ar (Either a b) c
```

Двудырочные интерфейсы

```
ar :: * → * → *
```

- категории

```
id    ::                      ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c   → ar a (b, c)
(***) :: ar a b → ar c d   → ar (a, c) (b, d)
```

- категории с копродуктами

```
(|||) :: ar a c → ar b c   → ar (Either a b) c
(+++) :: ar a b → ar c d   → ar (Either a c) (Either b d)
```

Двудырочные интерфейсы

```
ar :: * → * → *
```

- категории

```
id    ::                      ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c   → ar a (b, c)
(***) :: ar a b → ar c d   → ar (a, c) (b, d)
```

- категории с копродуктами

```
(|||) :: ar a c → ar b c   → ar (Either a b) c
(+++) :: ar a b → ar c d   → ar (Either a c) (Either b d)
```

- категории-моноиды

```
zeroA ::                      ar a b
(<+>) :: ar a b → ar a b   → ar a b
```

Двудырочные интерфейсы `ar :: * → * → *`

- категории

```
id    ::                      ar a a
(>>>) :: ar a b → ar b c   → ar a c
```

- категории с продуктами

```
(&&&) :: ar a b → ar a c   → ar a (b, c)
(***) :: ar a b → ar c d   → ar (a, c) (b, d)
```

- категории с копродуктами

```
(|||) :: ar a c → ar b c   → ar (Either a b) c
(+++) :: ar a b → ar c d   → ar (Either a c) (Either b d)
```

- категории-моноиды

```
zeroA ::                      ar a b
(<+>) :: ar a b → ar a b   → ar a b
```

- “классические” стрелки

```
arr   :: (a → b)           → ar a b
(>>^) :: ar a b → (b → c) → ar a c
```

Соответствие не по силе

- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`

Соответствие не по силе

- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`
- `(a → f b) → (a → f c) → a → f (b, c)`
  `ar a b → ar a c → ar a (b, c)`

Соответствие не по силе

- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`
- `(a → f b) → (a → f c) → a → f (b, c)`
  `ar a b → ar a c → ar a (b, c)`
- `a → f a`
  `ar a a`

Соответствие не по силе

- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`
- `(a → f b) → (a → f c) → a → f (b, c)`
  `ar a b → ar a c → ar a (b, c)`
- `a → f a`
  `ar a a`
- `(a → f b) → (a → f b) → a → f b`
  `ar a b → ar a b → ar a b`

Соответствие не по силе

- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`
- `(a → f b) → (a → f c) → a → f (b, c)`
  `ar a b → ar a c → ar a (b, c)`
- `a → f a`
  `ar a a`
- `(a → f b) → (a → f b) → a → f b`
  `ar a b → ar a b → ar a b`
- `a → f b`
  `ar a b`

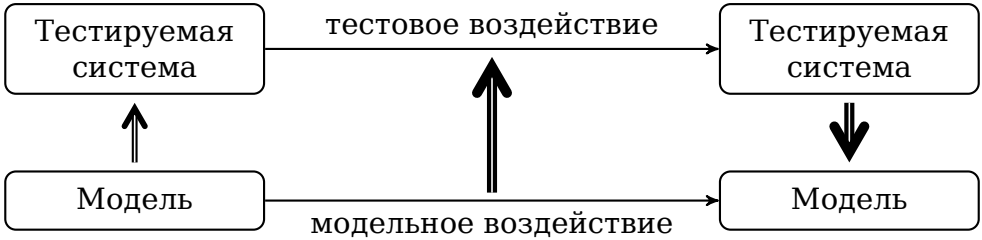
Соответствие не по силе

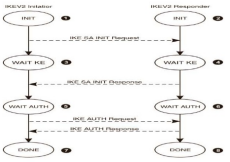
- `(a → f b) → (b → c) → a → f c`
  `ar a b → (b → c) → ar a c`
- `(a → f b) → (a → f c) → a → f (b, c)`
  `ar a b → ar a c → ar a (b, c)`
- `a → f a`
  `ar a a`
- `(a → f b) → (a → f b) → a → f b`
  `ar a b → ar a b → ar a b`
- `a → f b`
  `ar a b`
- `f (a → c) → f (b → c) → f (Either a b) → f c`
  `ar a c → ar b c → ar (Either a b) c`

Соответствие не по силе

- ```
  (a → f b) → (b → c) → a → f c
  ar a b    → (b → c) → ar a c
  ```
- ```
  (a → f b) → (a → f c) → a → f (b, c)
  ar a b    → ar a c    → ar a (b, c)
  ```
- ```
  a → f a
  ar a a
  ```
- ```
  (a → f b) → (a → f b) → a → f b
  ar a b    → ar a b    → ar a b
  ```
- ```
  a → f b
  ar a b
  ```
- ```
  f (a → c) → f (b → c) → f (Either a b) → f c
  ar a c    → ar b c    → ar (Either a b) c
  ```
- ```
  (a → f b) → (b → f c) → a → f c
  ar a b    → ar b c    → ar a c
  ```
  ```

Напоминание: медиаторы и сообщения





⇔

```
data SshState
  = Start
  | Versioned
  | Kexed { ...
```

⇔

```
msgToVars :: Payload -> [(Ins, Ins)]
msgToVars (Version ver) = [ (V "Clnt
msgToVars (KexInit KexData{..}) =
  [ (Vi "msgCode", C [20])
  , (V "Clnt.kexAlgos", commaSep kexA
  , (V "Clnt.servHostKeyAlgos", comma
  , (V "Clnt.encCTSAlgos", commaSep e
  , (V "Clnt.encSTCAlgos", commaSep e
  , (V "Clnt.macCTSAlgos", commaSep m
  , (V "Clnt.macSTCAlgos", commaSep m
  , (V "Clnt.compressCTSAlgos", comma
  , (V "Clnt.compressSTCAlgos", comma
```

⇔

Что такое, по сути, парсинг?

Что такое, по сути, парсинг?

Результат с побочным эффектом:

Что такое, по сути, парсинг?

Результат с побочным эффектом:

- имеет текущее состояние нераспарсенного

Что такое, по сути, парсинг?

Результат с побочным эффектом:

- имеет текущее состояние нераспарсенного
- может упасть с ошибкой

Что такое, по сути, парсинг?

Результат с побочным эффектом:

- имеет текущее состояние нераспарсенного
- может упасть с ошибкой
- продуцирует результат интересного типа

Что такое, по сути, парсинг?

Результат с побочным эффектом:

- имеет текущее состояние нераспарсенного
- может упасть с ошибкой
- продуцирует результат интересного типа

```
"12:a,b,c,d,e,f,g,h,k,l,m,n."
```

Что такое, по сути, парсинг?

Функция с побочным эффектом:

- имеет текущее состояние нераспарсенного
- может упасть с ошибкой
- продуцирует результат интересного типа
- имеет контекст

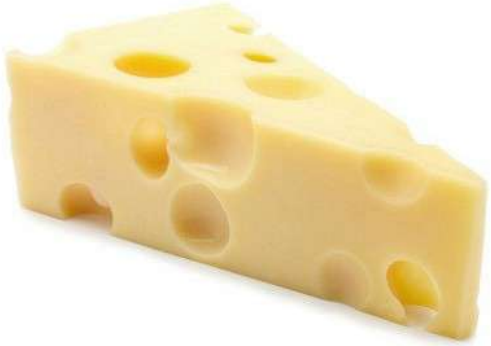
Что такое, по сути, парсинг?

Функция с побочным эффектом:

- имеет текущее состояние нераспарсенного
- может упасть с ошибкой
- продуцирует результат интересного типа
- имеет контекст

назревает вопрос...

Сколько дырок?



Жить можно и без дырок

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: а-ля `WriterT`
- имеет контекст: а-ля `ReaderT`

Жить можно и без дырок

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: а-ля `WriterT`
- имеет контекст: а-ля `ReaderT`

Жить можно… но недолго и закончится это грустно

Если в нём одна дыра...

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в функции слева

```
a → f b
```

Если в нём одна дыра...

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в функции слева

```
a → f b
```

+: функция фактически без контекста принимает простой вид
+: возможность использовать **do**-нотацию
-: для композиции с контекстом нужна монадическая композиция

Если в сыре много дыр…

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в дырке

```
ar a b
```

Если в сыре много дыр…

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: а-ля `StateT`
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в дырке

```
ar a b
```

+: простая последовательная композиция
+: куча дополнительных комбинаторов
-: непривычный синтаксис
-: непрозрачное протекание нераспарсенного в контекст
-: большинство простых парсеров имеют вид `ar () b`

Если в сыре много дыр...

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: в дырке
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: а-ля `ReaderT`
- имеет контекст: в дырке

```
ar a b
```

+: простая последовательная композиция
+: куча дополнительных комбинаторов
-: непривычный синтаксис
-: непрозрачное протекание нераспарсенного в контекст
-: большинство простых парсеров имеют вид `ar () b`

Если в сыре много дыр…

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: в дырке
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в дырке

```
p from ctx to
```

Если в сыре много дыр…

“Функция” с побочным эффектом:

- имеет текущее состояние нераспарсенного: в дырке
- может упасть с ошибкой: а-ля `ExceptT`
- продуцирует результат интересного типа: в дырке
- имеет контекст: в дырке

```
p from ctx to
```

+: упрощённая работа с нераспарсенным и контекстом
+ и -: как у двудырочного

На все деньги![1]

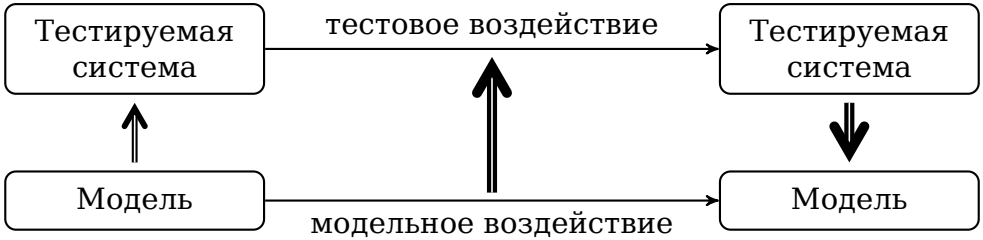
“Функция” с побочным эффектом:

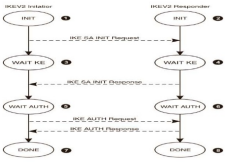
- имеет текущее состояние нераспарсенного: в дырке
- может упасть с ошибкой: в дырке
- продуцирует результат интересного типа: в дырке
- имеет контекст: в дырке

```
p e from ctx to
```

[1] https://hackage.haskell.org/package/boomerang-1.4.5.6/docs/Text-Boomerang-Prim.html

Напоминание: медиаторы и сообщения





⇔

```
data SshState
  = Start
  | Versioned
  | Kexed { ...
```

⇔

```
msgToVars :: Payload -> [(Ins, Ins)]
msgToVars (Version ver) = [ (V "Clnt
msgToVars (KexInit KexData{..}) =
  [ (Vi "msgCode", C [20])
  , (V "Clnt.kexAlgos", commaSep kexA
  , (V "Clnt.servHostKeyAlgos", comma
  , (V "Clnt.encCTSAlgos", commaSep e
  , (V "Clnt.encSTCAlgos", commaSep e
  , (V "Clnt.macCTSAlgos", commaSep m
  , (V "Clnt.macSTCAlgos", commaSep m
  , (V "Clnt.compressCTSAlgos", comma
  , (V "Clnt.compressSTCAlgos", comma
```

⇔

Парсинг сообщения: status quo

```
ssh2Payload src dst side =
  [ Vi "msgCode" # Len 1
  , mbopt $ Select (Vi "msgCode")
      [ Case [2] ignore
      , Case [5] serviceRequest
      , Case [6] serviceAccept
      , Case [20] [ sW "kexInitPayload" ## M keyExchangeInit ]
      , Case [21] newKeys
      , Case [30] dhInit
      , Case [31] dhReply ] ]
  where
    ignore         = [ V "ignoreData" ]
    serviceRequest = [ WithLen (BE 4) [ C $ ascii "ssh-userauth" ] ]

    ...
```

Парсинг сообщения: status quo

```
msgToVars (Version ver) = [ (V "Clnt.versionString", C $ ascii ver) ]
msgToVars (KexInit KexData{..}) =
  [ (Vi "msgCode", C [20])
  , (V "Clnt.kexAlgos",          commaSep kexAlgos)
  , (V "Clnt.servHostKeyAlgos", commaSep servHostKeyAlgos)
  , (V "Clnt.encCTSAlgos",       commaSep encCTSAlgos)
  , (V "Clnt.encSTCAlgos",       commaSep encSTCAlgos)
  , (V "Clnt.macCTSAlgos",       commaSep macCTSAlgos)
  , (V "Clnt.macSTCAlgos",       commaSep macSTCAlgos)
  , (V "Clnt.compressCTSAlgos", commaSep compressCTSAlgos)
  , (V "Clnt.compressSTCAlgos", commaSep compressSTCAlgos)
  , (V "Clnt.CTSLangs",          C $ ascii "")
  , (V "Clnt.STCLangs",          C $ ascii "")
  , (V "Clnt.firstKexPacketFollows", C [0]) ]
 ...
```

Парсинг сообщения: status quo

```
extractPayload = let getVal = MaybeT . extractValue . Prev in
  getVal (Vi "msgCode") >>= \case
   "\x14" -> do
     let unCommaSep x = B8.split ',' <$> getVal x
     kexAlgos_          <- unCommaSep $ V "Serv.kexAlgos"
     servHostKeyAlgos_ <- unCommaSep $ V "Serv.servHostKeyAlgos"
     encCTSAlgos_       <- unCommaSep $ V "Serv.encCTSAlgos"
     encSTCAlgos_       <- unCommaSep $ V "Serv.encSTCAlgos"
     macCTSAlgos_       <- unCommaSep $ V "Serv.macCTSAlgos"
     macSTCAlgos_       <- unCommaSep $ V "Serv.macSTCAlgos"
     compressCTSAlgos_ <- unCommaSep $ V "Serv.compressCTSAlgos"
     compressSTCAlgos_ <- unCommaSep $ V "Serv.compressSTCAlgos"
     pure $ KexInit $ KexData kexAlgos_ servHostKeyAlgos_ encCTSAlgos
 "\x15" ->
   pure NewKeys
...
```

Парсинг сообщения: светлое будущее

```
ssh2payload = version
          <+> ignore
          <+> serviceRequest
          <+> serviceAccept
          <+> keyExchangeInit
          <+> newKeys
          <+> dhInit
          <+> dhReply where
  version = _Version
        <$> constUtf8Text "SSH-2.0-"
         *> utf8Text `till` constChar '\n'

  ignore = _Ignore <$> constWord8 2 *> takeWhile (const True)

  serviceReq = _ServiceRequest
            <$> constWord8 5
             *> sizedByteSeq (word32 LittleEndian)
  ...
```

P.S. Пример `arrow do` с доски

```
data X = X Int [Char]
_X :: (X → (Int, [Char]), (Int, [Char]) → X) -- generated

number :: ar () Int
word32 :: Endianness → ar () Word32
seq    :: ar () a → ar Word32 [a]
char   :: ar () Char
dimap  :: (a → b, b → a) → ar () a → ar () b
ctxmap :: (a → b, b → a) → ar a b

parserX :: ar () X
parserX = proc () → do
  numField ← number    ⤙ ()
  len      ← word32 LE ⤙ ()
  chars    ← seq char  ⤙ len
             ctxmap _X ⤙ (numField, chars)
```

Интересное по теме

- Jeremy Gibbons, What You Needa Know About Yoneda (2019)
- Стрелки
  - Олег Нижников, После монад. Стрелочная и моноидальная композиция (2019)
  - John Hughes, Generalizing Monads To Arrows (1998)
  - John Hughes, Programming with Arrows (2005)
- John Hughes, How to specify it! A guide to writing properties of pure functions (2019)

Спасибо

Вопросы?