Adaptive Effect Handling in Frank

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

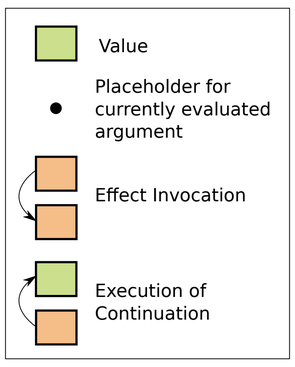
The Setting: Effectful Programming Language

Frank is a functional programming language with support for effect handlers, with the type system keeping track of the correct handling of effects.

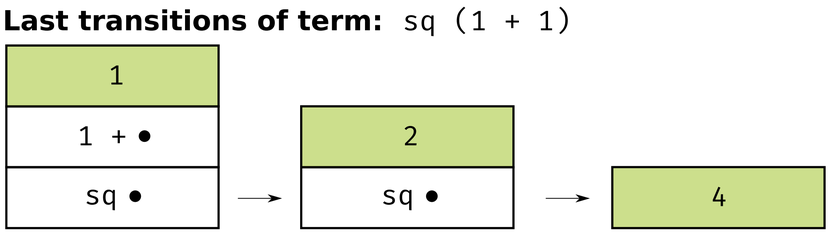
1) Without effects

The dynamic semantics of Frank can be observed via a stack machine.

- `sq` is a pure function computing the square of an int
- `sq: Int → Int`



Last transitions of term: `sq (1 + 1)`

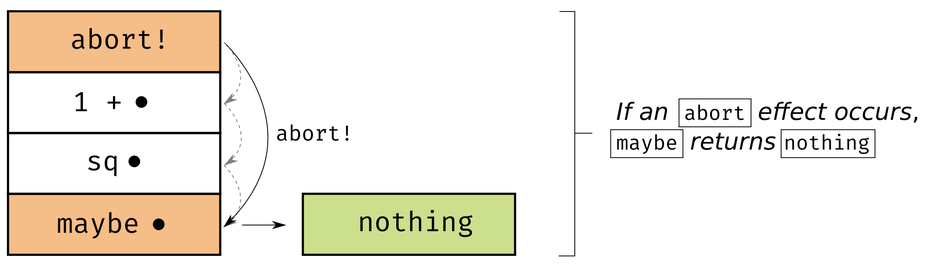


2) With exception effects

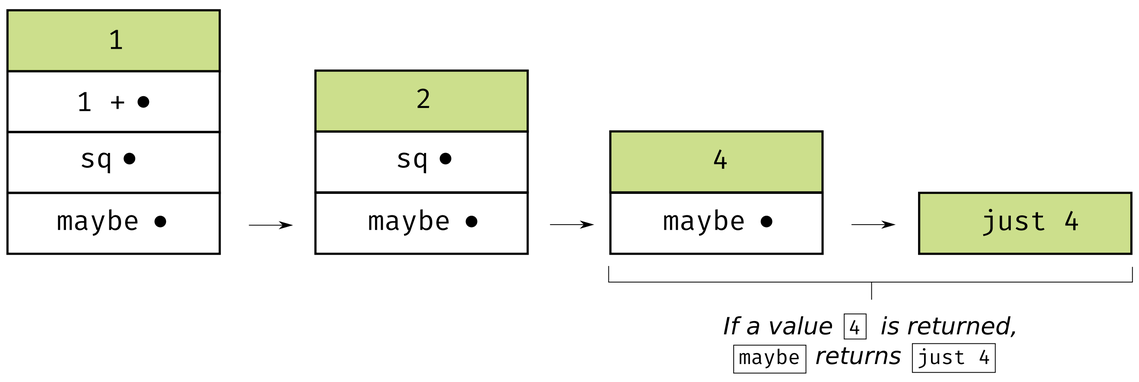
When an effect is invoked (here: `abort!`), the stack is traversed down until a matching handler is found (here: `maybe`). This handler then assumes control.

- `maybe` is a handler for `abort` effects, returning `Maybe X`
- `maybe: <abort>X → Maybe X`
- `maybe` assumes control in two cases:

i) An effect is invoked: `maybe (sq (1 + abort!))`



ii) A value is returned: `maybe (sq (1 + 1))`

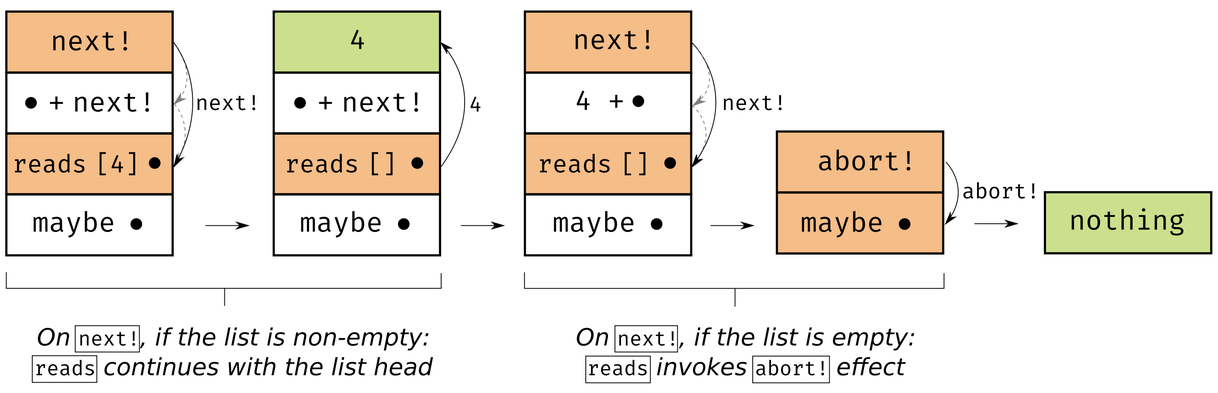


3) With exception + reader effects

When an effect is invoked (here: `next!`) and control is given to the handler (here: `reads`), the handler has access to the continuation and can run it. The continuation consists of the frames on top of the handler frame.

- `reads` is a handler (parameterised by a list) that generates effects on its own, namely `abort!` if the list is empty on an invocation of `next!`
- `reads xs: <next>X → [abort]X`

Last transitions of term: `maybe (reads [4] (next! + next!))`

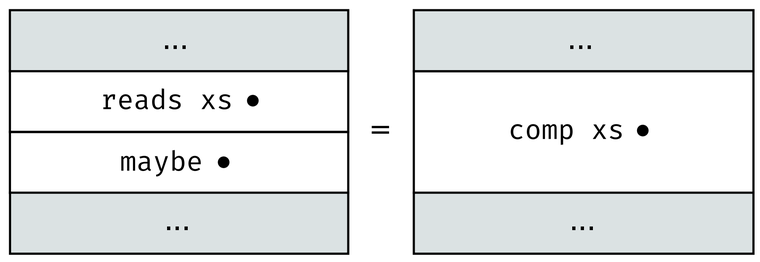


The Problem: Unintended Effect Capture

Consider the following composed operator.

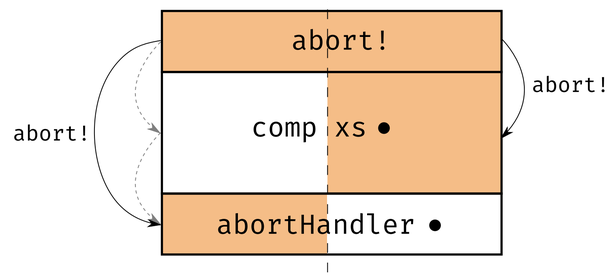
```
comp xs := maybe ∘ (reads xs)
```

It could have been used in the last example.



Now consider the following scenario: There is an abort effect intended for `excHandler` as part of some external exception handling mechanism.

Intention	What actually happens



- `comp` should handle `next!`
- `abortHandler` should handle `abort!`

- `comp` handles `next!` **and abort!**
- ⇒ The **external** `abort!` effects are captured by the **internal** `maybe` component.

When composing handlers, external effects interfere with internal handlers.

Adaptors as a Solution

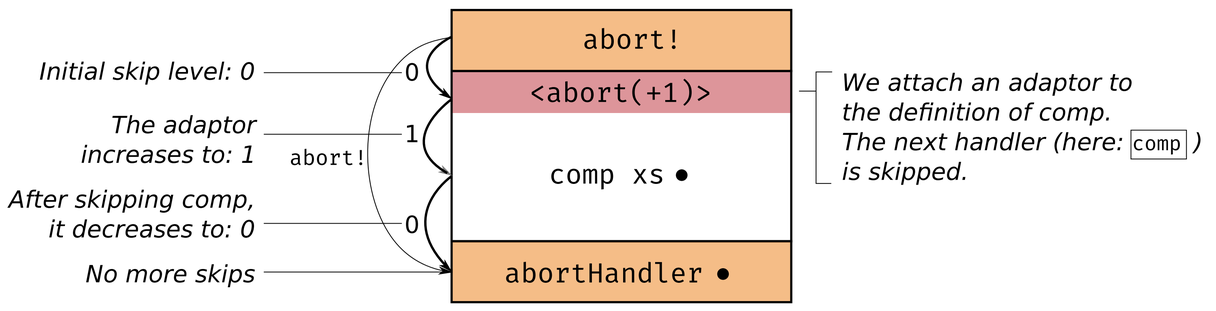
An adaptor reconfigures access to effect handlers.

Dynamic Semantics

Every effect request has a skip level that determines how many handlers on the way down should be skipped.

Intuition: An adaptor appears as a frame that redirects effect requests
Technically: An adaptor maps skip levels to adapted skip levels

Stack with adaptor included in the definition of `comp`



Static Semantics

Intuition: An adaptor rearranges the effects that are available in the current context
Technically: An adaptor maps effect contexts to adapted effect contexts

```
comp : List X → <next X> A → Maybe A
comp xs <m> = maybe (reads xs (<abort(+1)> m!))
```

Index notation →
Pattern matching notation (due to Sam Lindley) → `<abort(s x → s)>`
Short notation → `<abort>`

Adaptors

- **allow to reconfigure access to effects, thereby allowing the encapsulation of internal effects.**
- **are generally useful to compose effectful components.**

Related Work:

[1] Biernacki, Piróg, Polesiuk, Sieczkowski. 2018. Handle with care: relational interpretation of algebraic effects and handlers. POPL'18.
[2] Biernacki, Piróg, Polesiuk, Sieczkowski. 2019. Abstracting Algebraic Effects. POPL'19.
[3] Convent. 2017. Enhancing a Modular Effectful Programming Language. MSc Thesis, The University of Edinburgh
[4] Leijen. 2014. Koka: Programming with Row Polymorphic Effect Types. In MSFP (EPTCS), Vol. 153.
[5] Leijen. 2018. Algebraic Effect Handlers with Resources and Deep Finalization. Technical Report. Microsoft Research.
[6] Lindley, McBride, McLaughlin. 2017. Do be do be do. POPL'17.
[7] McBride. 2016. Shonky. https://github.com/pigworker/shonky
[8] Zhang, Salvaneschi, Beightol, Liskov, Myers. 2016. Accepting blame for safe tunneled exceptions. PLDI'16
[9] Zhang, Myers. 2019. Abstraction-safe effect handlers via tunneling. POPL'19.

Remarks: Much of this material is also part of a paper written together with co-authors Sam Lindley, Conor McBride and Craig McLaughlin which is currently under submission. Due to its simplicity, the author reused the composition of `maybe` with `reads` as a running example, originally proposed by Sam Lindley.