SE211 Projektni Zadatak - rexif2

Nikola Tasic 3698

04.06.2020.

Prolećni semetar, 2019/20

SE211: KONSTRUISANJE SOFTVERA

Projektni zadatak

rexif2

Ime i prezime: **Nikola Tasic**

Broj indeksa: **3698**

Datum izrade: **04.06.2020.**

Sadrzaj

Uvod

Razvoj ove biblioteke sluzi kao test performansi, ergonomicnosti i produktivnosti relativno novog jezika koji preti da ce zameniti neke od jezika koje smatramo industrijskim standardima. Takodje projekat je nastao kao zelja da se implementira program za sortiranje fotografija na osnovu EXIF podataka kao zamena za postojecu **Python** skriptu koja nizih performansi.

Cilj projekta

Primarni cilj ovog projekta je da se izradi performantna biblioteka za parsiranje podataka is TIFF fajlova. Sekundarni cilj da se na konkretnom primeru pokaze izrada programa po specfikaciji standarda neke, u ovom slucaju TIFF, tehnologije. Takodje jedna od svrha projekta je licni izazov u problematici koja barata low-level konceptima programiranja na nivou idividualnih bajtova.

Opis biblioteke

Biblioteka `rexif2` predstavlja implementaciju parsera **metadata** podataka (tagova) skladistenih u slikama sirovog formata. Sirov format slika koji ova biblioteka obradjuje spada u *TIFF revision 6.0* odnosno *RFC 2306*. Ova biblioteka je napisana za programski jezik **Rust** po ugledu na postojece biblioteke `exiv2` / `libexif` koje su predvidjene za programske jezike C i C++.

TIFF Format

TIFF format ili `Tagged Image File Format` predstavlja format za cuvanje fotografija bez kompresije kao i cuvanja dodatkih podataka o samoj fotografiji. Bas iz ovog razloga mnogi sirovi (RAW) formati koje koriste digitalni fotoaparati prate bas TIFF specifikaciju.

Struktura

TIFF fajlovi su organizovani u tri sekcije:

- Image File Header (IFH)
- Image File Directory (IFD)
- Bitmap podaci

Od ovih sekcija samo su prve dve neophodne, sto znaci da TIFF fajlovi ne moraju zapravo da imaju podatke o ikakvoj slici. Takvi fajlovi su moguci ali veoma neobicni i retki.

TIFF ume da bude veoma komplikovan format jer lokacije u samom fajlu gde se nalaze IFD sekcije i njihovi odgovarajuci bitmap podaci moze da varira. Jedino sto ima fiksnu lokaciju je zaglavlje (IFH) koji je uvek prvih 8 bajtova bilo kog TIFF fajla (sa par izuzetaka npr Nikon `.NEF` format). Svaki IFD i odgovarajuci bitmap podaci predtavljaju TIFF *podfajl* (*subfile*). Ne postoji ogranicenje koliko *podfajlova* moze da se nadje u jednom fajlu.

Svaki IFD sadzi jedan ili vise struktura koje nazivemo *tagovi*. Svaki tag je je velicine 12 bajtova i sadrzi neki deo informacije o TIFF fajlu (npr. duzinu ekspozicije). Tag moze da sadrzi bilo koji tip podataka i TIFF specifikacija definise preko 70 tagova koji predstavljaju razlicite informacije. Tagovi su redom rasporedjeni u svakom IFD-u. Napomena: Kompanije koje implementiraju TIFF standard cesto dodaju svoje tagove sa dodatnim informacijama tako da i o tome treba voditi racuna.

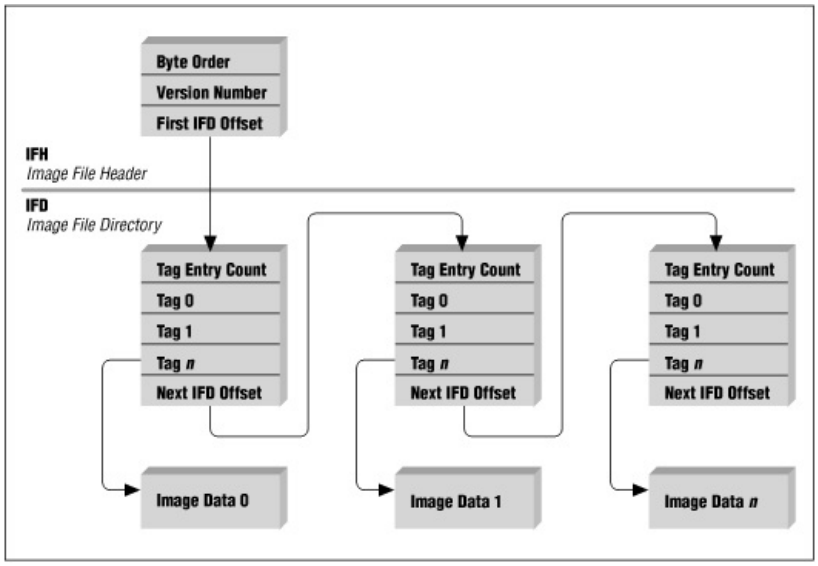

Fig. 1 Struktura TIFF fajla

U daljem tekstu cemo imati primere struktura u programskom jeziku C a kasnije implemetaciju u **Rust**-u.

Image File Header

```
// WORD - 16 bits = 2 bytes
// DWORD - 32 bits = 4 bytes
typedef struct _TiffHeader {
    WORD Identifier;   /* identifikator redosleta bajtova(endianness) */
    WORD Version;      /* TIFF verzija (uvek 2Ah) */
    DWORD IFDOffset;   /* odstupanje u bajtovima od prvog IFD-a */
} TIFHEAD;
```

Fig. 2 Zaglavlje u C-u

Identifikator moze da ima vrednost 4949h (II) ili 4D4Dh (MM). Ove vrednosti predtavljaju redosled podataka (bajtova) u TIFF fajlu - Intel format (little-endian) ili Motorola format (big-endian). Ove vrednosti su izabrane jer su identicne bez obzira na raspored.

Verzija je uvek 2Ah tj. decimalno 42 i ne menja se bez obziran na vrednost TIFF specifikacije. Obzirom na to da se ne menja vise predstavlja identifikator nego zapravo verziju. (*42 - answer to life, universe, etc*…).

Jedan od dva nacina kako TIFF fajl moze da pocinje:

```
49h 49h 2Ah 00h
```

ili

```
4Dh 4Dh 00h 2Ah
```

IFDOffset predstavlja 32-bitnu vrednost koja odredjuje gde se nalazi prvi Image File Directory.

Image File Directory

Image File Directory ili IFD predstavlja kolekciju informacija slicnjim zaglavlju i koristi se da opise bitmap podatke za koje je *zakacen*. Kao i zaglavlju on sadrzi podatke kao sto su na primer visina i sirina slike, kompresioni algoritam itd. Za razliku od fiksnog zaglavlja IFD je dinamican i moze biti ne samo bilo gde u fajlu nego ih moze biti i vise komada.

Fig. 3 Primer tri razlicite strukture IFD-a u TIFF fajlu

```
typedef struct _TifIfd
{
    WORD NumDirEntries;   /* broj tagova u IFD-u */
    TIFTAG TagList[];     /* lista tagova */
    DWORD NextIFDOffset;  /* odstupanje od sledeceg IFD-a */
} TIFIFD;
```

Fig. 4 IFD u C-u

`NumDirEntries` oznacava broj tagova u IFD-u i samim tim je ogranicena na 65,535. Svaki tag je struktura od 12 bajtova. NextIFDOffset predstavlja odstupanje od sledeceg IFD-a a u slucaju da ih nema vise to polje ima vrednost 00h.

Tagovi

Tag kao sto je pomenuto polje sa podacima u zaglavlju. Za razliku od podataka u zaglavljima koji su fiksne velicina tag moze da sadrzi podatke ili pokazivac ka lokaciji gde se nalaze podaci bilo koje velicine. Ova prilagodljivost tagova ima i svoju cenu. Tag koji treba da sadrzi podatke o samo jedom bajtu ipak mora biti velicine 12 bajtova.

```
typedef struct _TifTag
{
    WORD TagId;         /* identifikator */
    WORD DataType;      /* tip podatka */
    DWORD DataCount;    /* broj(kolicina) podataka */
    DWORD DataOffset;   /* odstupanje do mesta gde se nalaze podaci */
} TIFTAG;
```

Fig. 5 TIFF tag u C-u

`TagId` je numericka vredost informacije koju tag sadrzi. Iz tabele specifikacije mozemo procitati sta taj identifikator prestavlja.

`DataType` sadrzi numericku vrednost koja oznava koji tip podatka tag sadrzi. Neke od vrednost iz TIFF specifikacije su:

ID	TYPE	Description
1	BYTE	8-bit unsigned integer
2	ASCII	8-bit NULL-terminated string
3	SHORT	16-bit unsigned integer
4	LONG	32-bit unsigned integer
5	RATIONAL	Two 32-bit unsigned integers
6	SBYTE	8-bit signed integer
7	UNDEFINE	8-bit byte
8	SSHORT	16-bit signed integer
9	SLONG	32-bit signed integer
10	SRATIONAL	Two 32-bit signed integers
11	FLOAT	4-byte single-precision IEEE oating-point value
12	DOUBLE	8-byte double-precision IEEE oating-point value

Fig. 6 Lista tipova TIFF podataka

`DataCount` predstavlja broj podataka specificiranih od strane `DataType` polja.

`DataOffset` je 32-bitna (4 bajta) vrednost koja ukazuje na lokaciju podataka u TIFF fajlu koje taj tag opisuje u slucaju da je velicina podataka veca od 4 bajta, u suprotnom podaci se nalaze bas tu. Pakovanje podataka na ovoj lokaciji je jedna od optimizacija. Cesto se podaci ne nalaze ovde.

Neki od tipova podataka i njigove velicine se nalaze u sledecoj tabeli:

Tag Name	Tag ID	Tag Type
HalftoneHints	321	SHORT
HostComputer	316	ASCII
ImageDescription	270	ASCII
ImageHeight	257	SHORT or LONG
ImageWidth	256	SHORT or LONG
InkNames	333	ASCII
InkSet	332	SHORT
JPEGACTTables	521	LONG
JPEGDCTTables	520	LONG
JPEGInterchangeFormat	513	LONG

Fig. 7 TIFF tagovi

TIFF Fajl

```
00000000: 4d4d 002a 0000 0008 001a 00fe 0004 0000  MM.*............
00000010: 0001 0000 0001 0100 0004 0000 0001 0000  ................
00000020: 00a0 0101 0004 0000 0001 0000 0078 0102  ............x..
00000030: 0003 0000 0003 0000 0148 0103 0003 0000  .........H......
00000040: 0001 0001 0000 0106 0003 0000 0001 0002  ................
00000050: 0000 010f 0002 0000 0012 0000 0150 0110  ............P..
00000060: 0002 0000 000c 0000 0164 0111 0004 0000  .........d......
00000070: 0001 0001 e3f8 0112 0003 0000 0001 0001  ................
00000080: 0000 0115 0003 0000 0001 0003 0000 0116  ................
00000090: 0004 0000 0001 0000 0078 0117 0004 0000  .........x......
000000a0: 0001 0000 e100 011a 0005 0000 0001 0000  ................
000000b0: 0170 011b 0005 0000 0001 0000 0178 011c  .p..........x..
000000c0: 0003 0000 0001 0001 0000 0128 0003 0000  ...........(....
000000d0: 0001 0002 0000 0131 0002 0000 000a 0000  .......1........
000000e0: 0180 0132 0002 0000 0014 0000 018c 014a  ...2...........J
000000f0: 0004 0000 0003 0000 01a0 0214 0005 0000  ................
00000100: 0006 0000 01ac 02bc 0001 0000 0400 0000  ................
00000110: 01dc 8769 0004 0000 0001 0000 05f0 8825  ...i...........%
00000120: 0004 0000 0001 0001 e3e4 9003 0002 0000  ................
00000130: 0014 0000 05dc 9216 0001 0000 0004 0100  ................
00000140: 0000 0000 0000 0000 0008 0008 0008 0000  ................
00000150: 4e49 4b4f 4e20 434f 5250 4f52 4154 494f  NIKON CORPORATIO
00000160: 4e00 0000 4e49 4b4f 4e20 4433 3230 3000  N...NIKON D3200.
00000170: 0000 012c 0000 0001 0000 012c 0000 0001  ...,.......,....
00000180: 5665 722e 312e 3031 2000 0000 3230 3133  Ver.1.01 ...2013
00000190: 3a30 393a 3034 2031 333a 3131 3a34 3800  :09:04 13:11:48.
000001a0: 0002 c4f8 0002 c570 0002 c654 0000 0000  .......p...T....
000001b0: 0000 0001 0000 00ff 0000 0001 0000 0000  ................
000001c0: 0000 0001 0000 00ff 0000 0001 0000 0000  ................
000001d0: 0000 0001 0000 00ff 0000 0001 3c3f 7870  ............<?xp
000001e0: 6163 6b65 7420 6265 6769 6e3d 22ef bbbf  acket begin="...
000001f0: 2220 6964 3d22 5735 4d30 4d70 4365 6869  " id="W5M0MpCehi
```

Fig. 8 Nikon RAF format

Kao sto smo gore naveli svaki tiff fajl pocinje specificiranim zaglavljem. Iznad vidimo izlaz programa `xxd` koji nam pokazuje heksadecimalne i ASCII podatke iz kojih mozemo jasno da uvidimo da ovo zapravo jeste fajl TIFF formata i pocinje sa ‘motorola’ zaglavljem odnosno bajtovima `4d4d002a` . Takodje uocavamo `0x00000008` odmah zatim koji predstavlja offset od 8 bajtova do prvog IFD-a. Znajuci da je zamo TIFF zaglavje 8 bajtova dolazimo da zakljucka da odmah sledeci bajtovi predstavljaju broj tagova u prvom IFD-u kojih zaista ima `0x001a` tj 26.

Korisnički zahtevi

Funkcionalni zahtevi

- Biblioteka programeru korisniku treba da obezbedi metode koje iz njima prosledjenog falja vracaju strukturu koja reprezentuje `TIFF` fajl.
- Struktura koja reprezentuje `TIFF` fajl treba da sadrzi sve parsirane IFD-ove i Tag-ove iz `TIFF` fajla.
- Struktura koja reprezentuje `TIFF` fajl treba da sadrzi sve ne-parsirane Tag-ove tako da ih programer moze parsirati u zavisnosti od svojih potreba.
- Sve moguce greske treba razresiti idiomatski kroz sam jezik **Rust**.

Nefunkcionalni zahtevi

- Performanse biblioteke moraju biti bolje ili u najmanju ruku po brzini u rasponu od 10% odustpanja od brzine ekvivalentih biblioteke u jezicima `C/C++` . Jezik **Rust** se smatra visoko performantnim jezikom i to ova biblioteka treba da iskoristi.
- Bezbednost koriscenja treba da se ogleda u tome da korscenje biblioteke ni na koji nacin ne sme da utice na strukturu fajlova ili da ih na bilo koji nacin menja bez namere programera.

Klasni dijagram

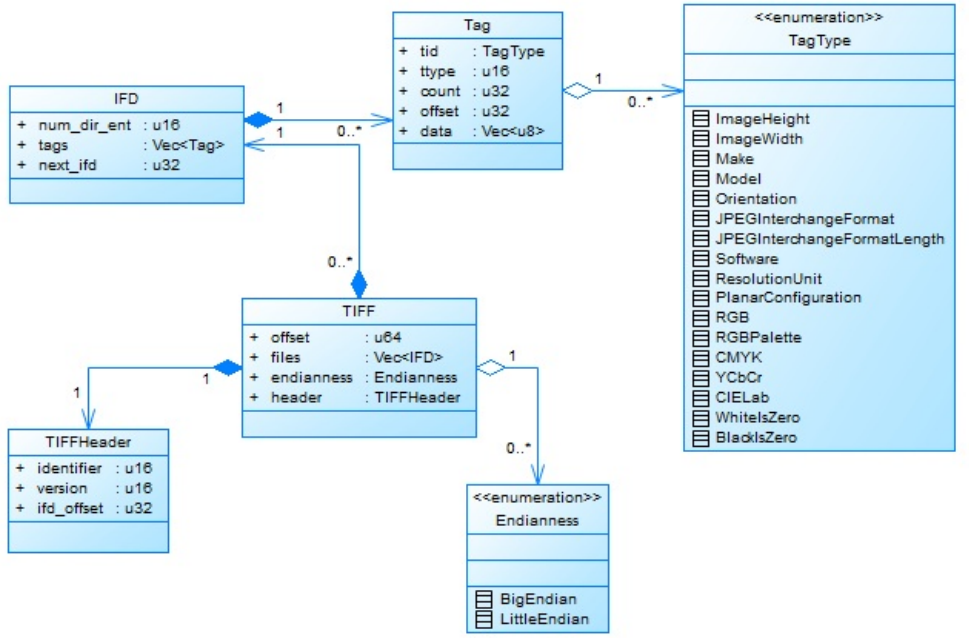

Fig. 9 Klasni dijagram TIFF strukture

Bez obzira na cinjenicu da **Rust** nije objektno-orijentisani jezik na slici imamo klasni dijagram koji predstavlja TIFF strukturu. U okviru `TagType` enumeracije prikazani su samo neki od nekoliko stotina standardnih i nestandardnih tipova TIFF tagova.

Tehinike i pravila kodiranja

Primeri komentara i `rustdoc`-a

Rust kao sam po sebi mocan jezik ima i mocan buildchain u vidu `cargo`-a. Cargo pored svih neophodnih funkcionalnosti za build-ovanje, dependency management i drugog ima mogucnost da generise automatsku `html` / `markdown` dokumentaciju na osnovu komentara u samom izvornom kodu. Pored osnovnih `//` komentara Rust ima i `///` komentari koji se u `cargo`-u interpretiraju kao dokumentacija.

```
/// Rust representation of TIFF IFD (Image File Directory)
///
/// C equivalent:
/// ```c
/// typedef struct _TifIfd
/// {
///     WORD NumDirEntries;
///     TIFTAG TagList[];
///     DWORD NextIFDOffset;
/// } TIFIFD;
/// ```
#[derive(Clone)]
pub struct Ifd {
    num_dir_ent: u16,
    pub tags: Vec<Tag>,
    next_ifd: u32,
}
```

Fig. 10 Primer rustdoc komentara/dokumentacije

```
/// Parses IFD struct from  a file
///
/// # Arguments
///
/// * `file_reader` - File reader from which we parse IFD
///
/// * `endianness` - Endianness to parse the data with
///
/// * `offset` - offset in bytes required for some vendor file types to skip arbitrary header eg. 160 for Fujifilm RAF
///
pub fn new(file_reader: &mut BufReader<&mut File>, endianness: Endianness, offset: u64) -> Self
```

Fig. 11 Primer rustdoc kometara metode

Lintovanje i `rustc`

Rust kompajler odnosno `rustc` je jedan od najsofisticiranijih kompajlera medju modernim jezicima. Njegov macro sistem je bukvalno jezik za sebe a borrow-checker je sistem koji omogucava type i memory safety do sada ne vidjen. Pored svega toga pruza i odlican linting i upozorenja za optimizaciju koda. Cesto se moze reci da ako se vas Rust kod uopste kompajlira znaci da se raditi bez greske. Alata za staticku analizu gotovo i da nema zbog samog dizajna jezika.

```
warning: unreachable pattern
   --> src/tagtype.rs:397:13
    |
397 |             6 => TagType::YCbCr,
    |             ^

warning: field is never used: `offset`
  --> src/tiff.rs:20:5
   |
20 |     offset: u64,
   |     ^^^^^^^^^^^
   |
   = note: `#[warn(dead_code)]` on by default

warning: method is never used: `size`
  --> src/tag.rs:85:5
   |
85 |     pub fn size(&self) -> usize {
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^

warning: variant is never constructed: `ExposureTime`
   --> src/tagtype.rs:226:5
    |
226 |     ExposureTime,
    |     ^^^^^^^^^^^^

warning: variant is never constructed: `FNumber`
   --> src/tagtype.rs:228:5
    |
228 |     FNumber,
    |     ^^^^^^^
```

Fig. 12 Neki od primera lintovanja

```
error[E0382]: use of moved value: `ifd`
  --> src/tiff.rs:48:29
   |
41 |         let mut ifd = Ifd::new(&mut file_reader, endianness.clone(), offset);
   |             ------- move occurs because `ifd` has type `ifd::Ifd`, which does not implement the `Copy` trait
...
45 |         tiff.files.push(ifd);
   |                         --- value moved here
...
48 |             tiff.files.push(ifd);
   |                             ^^^ value used here after move
```

Fig. 13 Primer veoma opsirnih poruka kod gresaka

```
error[E0308]: mismatched types
  --> src/tiff.rs:40:34
   |
40 |         let header = Header::new(header_buffer)?;
   |                                  ^^^^^^^^^^^^^
   |                                  |
   |                                  expected &[u8; 8], found array of 8 elements
   |                                  help: consider borrowing here: `&header_buffer`
   |
   = note: expected type `&[u8; 8]`
              found type `[u8; 8]`
```

Fig 14. Poruke borrow-checkera

Testiranje

Za testiranje u **Rust** programskom jeziku je dovoljno koristiti vec pomenuti `cargo` .

Definisanje slucajeva testiranja se vrsi preko macro direktiva u samom izvornom kodu.

```
#[cfg(test)]
mod tests {
    use crate::header::Header;

    /// test slucajevi
}
```

Fig 15. Definisanje modula za testiranje

Jedinicno testiranje

```
#[test]
pub fn test_ii() {
    let buf = [0x49, 0x49, 0x2A, 0x00, 0x00, 0x00, 0x00, 0x00];
    let ii_header = Header::new(&buf).expect("Failed to parse II header.");
    println!("Parsed identifier of value {:?} from buffer {:?}", ii_header.identifier, buf);
    assert_eq!(ii_header.identifier, 0x4949);
    assert_eq!(ii_header.version, 42);
}

#[test]
pub fn test_zero_offset_le() {
    let buf = [0x49, 0x49, 0x2A, 0x00, 0x00, 0x00, 0x00, 0x00];
    let header = Header::new(&buf).expect("Failed to parse header.");
    assert_eq!(header.ifd_offset, 0x00000000);
    assert_eq!(header.version, 42);
}

#[test]
pub fn test_zero_offset_be() {
    let buf = [0x4D, 0x4D, 0x00, 0x2A, 0x00, 0x00, 0x00, 0x00];
    let header = Header::new(&buf).expect("Failed to parse header.");
    assert_eq!(header.ifd_offset, 0x00000000);
    assert_eq!(header.version, 42);
}
```

Fig. 16. Primer jedinicnih slucajeva testiranja

```rust
#[test]
pub fn test_invalid_version_buf() {
    let buf = [0x4D, 0x4D, 0x43, 0x29, 0x00, 0x00, 0x72, 0x2A];
    let header = Header::new(&buf);
    match header {
        Ok(_) => { assert!(false, "Parsing of invalid header didn't throw an error") }
        Err(_) => { assert!(true, "Parsing of invalid header failed") }
    }
}

#[test]
pub fn test_invalid_identifier_buf() {
    let buf = [0x47, 0x52, 0x2A, 0x00, 0x00, 0x00, 0x72, 0x2A];
    let header = Header::new(&buf);
    match header {
        Ok(_) => { assert!(false, "Parsing of invalid header didn't throw an error") }
        Err(_) => { assert!(true, "Parsing of invalid header failed") }
    }
}
```

Fig. 17 Primer jedinicnih slucajeva testiranja

Pokretanje testova:

```
nik rexif2 master $ cargo test

running 9 tests
test header::tests::test_ii ... ok
test header::tests::test_invalid_identifier_buf ... ok
test header::tests::test_mm ... ok
test header::tests::test_invalid_version_buf ... ok
test header::tests::test_offset_be ... ok
test header::tests::test_offset_le ... ok
test header::tests::test_zero_offset_be ... ok
test header::tests::test_zero_offset_le ... ok
test tests::test ... ok

test result: ok. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Fig. 18 Rezultati testiranja

Bibliografija

- P. Bourke, *TIFF File Format Summary*, Poseceno: 08. Jun, 2020, [Online], Dostupno na: paulbourke.net/dataformats/tiff/tiff_summary.pdf
- Network Working Group, *Tag Image File Format (TIFF) - F Profile for Facsimile*, Mart 1998, Poseceno: 08. Jun, 2020, [Online], Dostupno na: tools.ietf.org/html/rfc2306
- Rust Lang, *std::doc*, 2020, Poseceno: 08. Jun, 2020, [Online], Dostupno na: www.rust-lang.org
- S. Klabnik and C. Nichols, *The Rust Programming Language*, San Francisco, CA, USA: No Starch Press, 2018, ISBN 9781593278281