VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Detekcia video kontajnerov v dátovom streame

20. februára 2011

Michal Kučiš
xkucis00@stud.fit.vutbr.cz

pod vedením doc. Adama Herouta
herout@fit.vutbr.cz

Tato Práce vznikla za podpory projektu MŠMT CZ.1.07/2.3.00/09.0067
TeamIT - Budování konkurenceschopných výzkumných týmu pro IT

Abstrakt

Tento článok popisuje algoritmy, ktoré detekujú najrozšírenejšie video kontajnery v súbore a v dátovom streame. Na úvod je uvedený zoznam funkcií a príkazov použitých v popisovaných algoritmoch. Ďalej sú uvedené algoritmy, ktoré detekujú video kontajnery na základe bytov nachádzajúcich sa na začiatku súboru. Ak nie je možné zistiť začiatok súboru, je možné použiť algoritmy uvedené ako parsery. Tieto algoritmy umožňujú detekovať video kontajner na základe bytov, ktoré sa nachádzajú v strede súboru. Výhoda týchto algoritmov je, že nie je nutné zistovať začiatok súboru. Výrazná vevýhoda je, že je nutné spracovať väčšie množstvo dát ako v algoritmoch detekujúcich hlavičky.

Obsah

1 Zoznam použitých príkazov a funkcií

CMP(COUNT_BITS, BITS)

- príkaz získa prvých COUNT_BITS bitov zo streamu a porovná ich s hodnotou BITS; následne odstráni zo streamu COUNT_BITS bitov
- ak porovnanie neuspeje, test vráti chybu (zavolá sa príkaz ERROR())
- ak porovnanie uspeje, test pokračuje ďalším príkazom
- ak streame bude obsahovať 0x464C561234, príkaz CMP(24,0x464C56) uspeje. Po zavolaní príkazu, bude stream obsahovať 0x1234
- test CMP(24, 0x464C56) je ekvivalentný s príkazovým blokom {CMP(8,0x46); CMP(8,0x4C); CMP(8,0x56);}

SKIP(COUNT_BITS)

- príkaz odstráni zo streamu prvých COUNT_BITS bitov

GET(COUNT_BITS)

- funkcia vráti POCET_BITOV zo streamu, následné odstráni prvých COUNT_BITS bitov zo streamu

ERROR()

- ukončí test a vráti chybu

OK()

- ukončí test a vráti OK

GET_STREAM_BEGIN()

- príkaz vráti aktuálnu pozíciu v streame
- viz. SET_STREAM_BEGIN

SET_STREAM_BEGIN(POS)

- nastaví pozíciu v streame
- ukážka použitia funkcií GET_STREAM_BEGIN a SET_STREAM_BEGIN:

```
// stream obsahuje: 0x12345678
int data = GET_STREAM_BEGIN ();
// stream obsahuje: 0x12345678
SKIP(8);
// stream obsahuje: 0x345678
SET_STREAM_BEGIN (data);
// stream obsahuje: 0x12345678
```

2 FLV hlavička

- testuje sa len začiatok súboru

test:

```
CMP (24,0x464C56);
CMP (8, 1);
CMP (5, 0);
SKIP(1);
CMP (1, 0);
SKIP(1);
CMP (32,9);
CMP (32,0);
CMP (2, 0);
SKIP(1);
int type = GET(5);
switch (type)
{
case 8:
case 9:
case 18:
break;
default:
ERROR();
}
SKIP(32);
CMP(24,0);
OK();
```

3 MKV hlavička

- testuje sa len začiatok súboru

test:

```
CMP(16, 0x1A45);
CMP(16, 0xDFA3);
CMP(1, 1);
int count = GET(5)-1;
CMP(2, 3);
for (int i=0; i¡count; i++)
{
    CMP (8, 0x42);
    int value = GET(8);
    switch (value)
    {
        case 0x86:
        case 0xF7:
        case 0xF2:
        case 0xF3:
        case 0x87:
        case 0x85:
            CMP(8,0x81);
            SKIP(8);
            break;
        case 0x82:
            CMP(8,0x88);
            CMP(8,'m');
            CMP(8,'a');
            CMP(8,'t');
            CMP(8,'r');
            CMP(8,'o');
            CMP(8,'s');
            CMP(8,'k');
            CMP(8,'a');
            break;
        default:
            ERROR();
        }
    }
OK();
```

4 AVI hlavička

- testuje sa len začiatok súboru

test:

```
CMP(8, 'R');
CMP(8, 'I');
CMP(8, 'F');
CMP(8, 'F');
SKIP(32);
CMP(8, 'A');
CMP(8, 'V');
CMP(8, 'I');
CMP(8, ' ');
CMP(8, 'L');
CMP(8, 'I');
CMP(8, 'S');
CMP(8, 'T');
SKIP(32);
CMP(8, 'h');
CMP(8, 'd');
CMP(8, 'r');
SKIP(8);
CMP(8, 'a');
CMP(8, 'v');
CMP(8, 'i');
CMP(8, 'h');
OK();
```

5 MPEG hlavička

- testuje sa len začiatok súboru

test:

```
CMP(32, 0x000001BA);
SKIP(64);
CMP(32, 0x000001BB);
int shSize = GET(16);
if (shSize < 8)
    ERROR();
if (shSize > 0x1f)
    ERROR();
OK();
```

6 TS hlavička

- testuje sa len začiatok súboru

test:

```
CMP(8, 0x47);
CMP(1 ,0);
SKIP(1);
SKIP(1);
int PID = GET(13);
SKIP(2);
int bAFExist = GET(1);
SKIP(1);
SKIP(4);
if (bAFExist)
{
    int AFSize = GET(8);
    SKIP(AFSize*8);
}
if (PID==0)
{
    int table_id = GET(8);
    switch(table_id)
    {
        case 0:
        case 2:
        case 8:
        case 34:
        OK();
    default:
        ERROR();
    }
}
ERROR();
```

7 MP4 hlavička

- testuje sa len začiatok súboru

test:

```
bool testMP4X()
{
    CMP(8, 'm');
    CMP(8, 'p');
    CMP(8, '4');
    int c = GET(8);
    if (c == '2')
        return true;
    if (c == '1')
        return true;
    return false;
}

CMP(24, 0);
int count = GET(6);
CMP(2, 0);
CMP(8, 'f');
CMP(8, 't');
CMP(8, 'y');
CMP(8, 'p');
if(testMP4X())
    OK();
CMP(24, 0);
SKIP(8);
for (int i = 4; i < count; i++)
    if (testMP4X())
        OK();
ERROR();
```

8 AVI parser

test by mal spĺňať nasledujúce podmienky:

- je nutné preskočiť minimálne 130,000 bytov
- test má začať od 340,000 rôznych pozícii (pre väčšinu súborov stačí 30.000)
- jeden test sa má vykonať nad 800,000 bytoch (pre väčšinu súborov stačí COUNT_CYCLES=1)

test:

```
const int MAX_SIZE_SEGMENT = 340000;
while (COUNT_CYCLES--)
{
    int stream_begin=GET_STREAM_BEGIN();
    CMP(8, '0');
    char c1 = GET(8);
    if ((c1 < '0') || (c1 > '9'))
        ERROR();
    char c2 = GET(8);
    char c3 = GET(8);
    if (! ((c2 == 'w' && c3 == 'b') || (c2 == 'd' && c3 == 'c')
                || (c2 == 'd' && c3 == 'b')))
        ERROR();
    int sizeofSkip = 0;
    for (int i = 0; i < 4; i++)
        sizeofSkip += GET(8)<<8*i;

    if (sizeofSkip > MAX_SIZE_SEGMENT)
        return false;
    if (sizeofSkip & 1)
        sizeofSkip += 1;

    SET_STREAM_BEGIN(stream_begin + sizeofSkip + 8);
}
```

9 FLV parser

test by mal spĺňať nasledujúce podmienky:

- nie je nutné nič preskakovať
- test má začať od 40,000 rôznych pozícii
- jeden test sa má vykonať nad 80,000 bytoch (pre väčšinu súborov stačí COUNT_CYCLES=1)

test:

```
const int MAX_SIZE_SEGMENT = 40000;
int nsizePrevTag = -1;
int nsizeThisTag = -1;
while (COUNT_CYCLES--)
{
    int stream_begin=GET_STREAM_BEGIN();
    nsizePrevTag = GET(32);
    if ((nsizeThisTag!=-1) && (nsizePrevTag!=nsizeThisTag))      ERROR();
    CMP (2, 0);
    SKIP (1);
    int type = GET (5);
    if (! ((type==8) || (type==9) || (type==18)))      ERROR();
    nsizeThisTag = GET(24)+11;
    if (nsizeThisTag > MAX_SIZE_SEGMENT)
        ERROR();
    SKIP(32);
    CMP (24,0x000000);
    if (type==9)
    {
        int type = GET (4);
        int codec = GET (4);
        if ((type < 1) || (type > 5))      ERROR ();
        if ((codec < 2) || (codec > 7))      ERROR ();
        if (codec==7)
        {
            int avctype = GET (8);
            int comp = GET (24);
            if (avctype > 2)
                ERROR ();
            if ((avctype!=1) && (comp!=0))
                ERROR ();
         }
    }
    SET_STREAM_BEGIN (stream_begin + nsizeThisTag + 4);
}
```

10 MPG parser

test by mal spĺňať nasledujúce podmienky:

- nie je nutné nič preskakovať
- test má začať od 4,100 rôznych pozícii
- jeden test sa má vykonať nad 8,300 bytoch (pre väčšinu súborov stačí COUNT_CYCLES=1)

test:

```
const int MAX_SEGMENT_SIZE=4100;
const int MIN_SEGMENT_SIZE=2000;
while (COUNT_CYCLES--)
{
    int stream_begin = GET_STREAM_BEGIN();
    int cod = GET (32);
    if (cod != 0x000001BA)
        ERROR();
    __int64 SCR = 0;
    SKIP (2);
    SCR |= GET(3) << 30;
    SKIP (1);
    SCR |= GET(15) << 15;
    SKIP (1);
    SCR |= GET (15) << 0;
    SKIP (1);
    int SCREx = GET (9);
    SKIP (1);
    SCR *= 300;
    SCR |= SCREx;
    SKIP(16);
        int tmp_stream_begin = GET_STREAM_BEGIN();
    int testedByte = GET(16) & 0x00ff;
    if (testedByte)
    {
        SKIP(8);
        int step = GET(3);
        SKIP(5);
        SKIP(8*step);
    }
    else
        SET_STREAM_BEGIN (tmp_stream_begin);
```

pokračovanie:

```
    cod = GET(32);
    if(          cod == 186+256
                   || cod == 187+256
                   || cod == 188+256
                   || cod == 189+256
                   || cod == 190+256
                   || cod == 191+256
                   || (cod >= 192+256 && cod <= 223+256)
                   || (cod >= 224+256 && cod <= 239+256)
                   || (cod >= 240+256 && cod <= 256+255)
                   )
        DO_NOTHING;
    else
        ERROR();

    // SCR testovanie je velmi účinné na filtrovanie..
    // Ak COUNT_CYCLES==1, nasledujúci kód nemá zmysel :)
    if (cod >= 224+256 && cod <=239+256)
    {
        int index = cod - (224+256);
        if (SCR <= inout_prevVideoSCR)
            ERROR();
        inout_prevVideoSCR = SCR;
    }

    // next_header:
    // - obsahuje nedeterministicky získanú hodnotu
    //       - hodnota je medzi MIN_SEGMENT_SIZE a MAX_SEGMENT_SIZE
    // - v rôznych mpg súboroch je táto hodnota rôzna
    // - v niektorých súboroch mpg sá táto hodnota mení aj v rámci jedného súboru
    int next_header = ???;
    SET_STREAM_BEGIN (stream_begin+next_header);
}
```

11 TS parser

test by mal spĺňať nasledujúce podmienky:

- nie je nutné nič preskakovať
- test má začať od 188 rôznych pozícii
- jeden test sa má vykonať nad 600 bytoch

test:

```
while (COUNT_CYCLES–)
{
    int stream_begin = GET_STREAM_BEGIN();
    int PID,bAFExist;
    // obvykle stačí testovať len 0x47 a zbytok preskočiť...
    // len potom treba aj väčšiu hodnotu COUNT_CYCLES
    CMP (8, 0x47);
    if (GET(1))
        ERROR ();
    SKIP(2);
    PID = GET(13);
    SKIP(2);
    bAFExist = GET(1);
    SKIP(5);
    if (bAFExist)
    {
        int AFSize = GET (8);
        SKIP (AFSize*8);
    }
    if (PID==0)
    {
        int table_id = GET(8);
        switch (table_id)
        {
            case 0:
            case 2:
            case 8:
            case 34:
                break;
            default:
                ERROR ();
        }
    }
    SET_STREAM_BEGIN (stream_begin+188);
}
```

12 MKV parser

test by mal spĺňať nasledujúce podmienky:

- je nutné preskočiť minimálne 8,000,000 bytov
- test má začať od 80,000 rôznych pozícii
- jeden test sa má vykonať nad 160,000 bytoch

popis:

- v matroske sú data metadata organizované do elementov
- elementy vytvárajú hierarchiu
- každý element má parametre:
  - CODE - identifikátor elementu (v súbore sa môže vyskytovať viacero elementov s rovnakou hodnotou CODE)
  - LVL - úroveň, na ktorej sa element môže nachádzať; vyplýva z hodnoty CODE
  - SIZE - veľkosť dát, element obsahuje
  - TYP - vyplýva z CODE; viz. nižšie,
- tieto parametre sme si zaviedli na zjednodušenie ďalšieho popisu, v popise EBML tieto parametre neexistujú (v EBML je MKV zakódované)
- fyzicky je element na disku tvaru: | CODE | SIZE | data |
  - veľkosť položiek CODE/SIZE je zakódovaná v samotnej položke:
    - ak je 1. bit nenulový, položka je veľká 1 byte
    - ak je 1. bit nulový a 2. nenulový, položka je veľká 2 byte
    - ak je 1. a 2. bit bit nulový a 3. bit je nenulový, položka je veľká 3 byte
    - ...
    - príklad:
      CODE=0x1f43b675 ma 3 bity nulové a 4. nenulový => položka CODE je dlhá 4 byty
  - dobrý popis je http://www.matroska.org/technical/specs/index.html
- Pre zjednušenie si zavedieme 3 typy elementov:
  - MASTER
    - ak je element LVL-u n, nasledujúci element musí byt LVL-u n+1
    - na disku sú elementy usporiadané:
      | CODE_MASTER | SIZE | CODE_??? | SIZE_??? | ...
    - predpokladajme, že v streame sú bity 0x82ff8781. Zo streamu môžme získať nasledujúce informácie:
      - 0x82 - kód elementu, o ktorom vieme, že je typu MASTER
      - 0xff - element je veľký 7f

        - 0x87 - kód ďalšieho elementu
        - 0x81 - veľkosť ďalšieho elementu
    - VALUE
        - dátový element, ktorý obsahuje maximálne 80,000 bytov dat
        - na disku je tvaru: | CODE_VALUE | SIZE | data |
        - ak je tento element LVL-u n, nasledujúci element musí byť LVL-u max n
        - predpokladajme, že v streame sú bity 0x878152a3. Zo streamu môžme získať nasledujúce informácie:
            - 0x87 - kód elementu
            - 0x81 - veľkosť elementu
            - 0x52 - data elementu, ktoré nás nezaujímajú=>preskakujeme ich
            - 0xa3 - kód ďalšieho elementu
    - SVALUE
        - typ je odvodený z VALUE
        - element typu SVALUE môže obsahovať maximálne 8 bytov dat (ostatné vlastnosti sú rovnaké ako v elemente typu VALUE)

- elementy, ktoré sú známe (a majú sa testovať). Elementy s inými CODEami vyvolajú chybu:
    - MASTER(0x1f43b675,1);
    - MASTER(0x5854,2);
    - MASTER(0xa0,2);
    - MASTER(0x75a1,3);
    - MASTER(0xa6,4);
    - VALUE (0xa3,2);
    - VALUE (0xa1,3);
    - SVALUE(0xE7,2);
    - SVALUE(0xa7,2);
    - SVALUE(0xab,2);
    - SVALUE(0x58d7,3);
    - SVALUE(0x9b,3);
    - SVALUE(0xfa,3);
    - SVALUE(0xfb,3);
    - SVALUE(0xee,5);
    - SVALUE(0xa5,5);

    popis elementov je tvaru TYP(CODE,LVL)

- príklad, ako spracovávať stream:
    - v streame mame bity 0x58548458d78105

- 1. bit je nulový, 2.bit je nenulový=>CODE je dlhý 2 byty =>CODE je 0x5854=>element s kódom 0x5854 poznáme ako element typu MASTER
- po odstránení CODE zo streamu, v streame ostane: 0x8458d78105
- 1. bit je nenulový=>položka SIZE je dlhá 1 byte =>položka SIZE je 0x84=>v položke SIZE je uložená hodnota 0x04=>data priradene elementu sú dlhé 4 byty
- po odstránení SIZE zo streamu v streame dostávame: 0x58d78105
- keďže sme spracovali element typu MASTER LVL 2, očakávame, že nasledujúci je LVL-u 3
- zo streamu získame 0x58d7=>element je LVL-u 3 (to je v poriadku) a je typu SVALUE
- zo streamu ďalej získame SIZE 0x81, v nej je uložená hodnota 0x01=>data tohto elementu sú dlhé 1 byte
- preskakujeme 1 byte
- týmto sme vyčerpali celý stream... všetky kontroly prebehli v poriadku a prehlasujeme, že sme detekovali mkv súbor