The Hoffman Tutorial

Brent Baccala

August 29, 2012

Hoffman is a program to solve chess endgames using retrograde analysis. A retrograde analysis program is much different from a conventional computer chess programs. Retrograde analysis is only useful in the endgame, runs very slowly, and produces enormous amounts of data. Its great advantage lies in its ability to completely solve the endgame. In a very real sense, a retrograde engine has no "move horizon" like a conventional chess engine. It sees everything. For those not up on Americana, the program is named after Trevor Hoffman, an All Star baseball pitcher who specializes in "closing" games. It was written specifically for The World vs. Arno Nickel game.

The first thing to understand is that Hoffman uses XML control files to govern its operation. Hoffman comes with an `xml` directory containing a number of these control files. I'll illustrate how Hoffman works using these examples. For example, Figure 1 shows about the simplest possible XML control file, for the king vs. king endgame, contained in the file `xml/kk.xml`.

What does this all mean? Well, the first two lines identify this as a Hoffman XML tablebase file; copy them verbatim at the top of any new Hoffman control file. The third line illustrates XML's comment format; anything between `<!--` and `-->` is a comment, in this case the identity of the Perl script that created this file. The bulk of the file is contained between the two lines `<tablebase>` and `</tablebase>`. `tablebase` is a simple example of an XML *element*, which come in basically two flavors: the simple kind and the complex kind. The simple kind, like, say, `piece`, have only a single tag that ends with a slash. The complex kind have both a beginning and an ending tag. The beginning tag does not have a slash, and the ending tag is identical to the beginning tag except that it starts with a slash (like `tablebase`). Complex elements allow you to put things between the starting and ending tags. Follow my lead; it'll start to make sense soon.

The `piece` elements are probably self-explanatory. Notice that I didn't specify where on the board the kings were. That's because we're going to compute results for every chess position possible with these two pieces. Their movements are completely unrestricted — they can be anywhere on the board. Later we'll see more complex piece elements that restrict where on the board the pieces can be.

Finally, we come to `output`, which tells the program where to put the output tablebase, in this case, into a file called

---

```xml
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Created by genctlfile.pl -->

<tablebase>
   <piece color="white" type="king"/>
   <piece color="black" type="king"/>
   <output filename="kk.htb"/>
</tablebase>
```

Figure 1: `kk.xml`

```
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Created by genctlfile.pl -->

<tablebase>
    <piece color="white" type="king"/>
    <piece color="black" type="king"/>
    <piece color="white" type="queen"/>
    <futurebase filename="kk.htb"/>
    <output filename ="kqk.htb"/>
</tablebase>
```

Figure 2: `kqk.xml`

`kk.htb`. You can rename the resulting tablebase file as you wish; the tablebase isn't tied to a particular filename.

Now you put all this into a file called something like `kk.xml`, (or just copy it from the `xml` directory) and run Hoffman like this:

```
C> hoffman -g kk.xml
Initializing tablebase
Checking futuremoves...
All futuremoves handled under move restrictions
Intra-table propagating
Pass 0 complete; 840 positions processed
Pass 1 complete; 0 positions processed
C>
```

There isn't much to see, of course. The "-g" option meant "generate". King vs king is nothing more than figuring out the difference between illegal positions and draws. But this information is important, because it's needed to back propagate from the (slightly) more complex three piece endgames, like the king and queen endgame whose XML configuration is shown in Figure 2.

Notice the new "futurebase" line. This tells Hoffman where to get the information about what happens when the black king can capture the white queen, because a tablebase contains information for *exactly* the piece configuration it is set up for — nothing more, nothing less.

So put all this in a file called `kqk.xml`, make sure the `kk.htb` file from the first run is present, and run Hoffman again:

```
C> ./hoffman -g -o kqk.htb kqk.xml
Initializing tablebase
Back propagating from 'kk.htb'
Checking futuremoves...
All futuremoves handled under move restrictions
Intra-table propagating
Pass 0 complete; 131516 positions processed
Pass 1 complete; 364 positions processed
Pass 2 complete; 2448 positions processed
Pass 3 complete; 1352 positions processed
```

```
Pass 4 complete; 5012 positions processed
Pass 5 complete; 2956 positions processed
Pass 6 complete; 9064 positions processed
Pass 7 complete; 7480 positions processed
Pass 8 complete; 19964 positions processed
Pass 9 complete; 14144 positions processed
Pass 10 complete; 26164 positions processed
Pass 11 complete; 25484 positions processed
Pass 12 complete; 32064 positions processed
Pass 13 complete; 39908 positions processed
Pass 14 complete; 32104 positions processed
Pass 15 complete; 54052 positions processed
Pass 16 complete; 15000 positions processed
Pass 17 complete; 43800 positions processed
Pass 18 complete; 2680 positions processed
Pass 19 complete; 11300 positions processed
Pass 20 complete; 8 positions processed
Pass 21 complete; 56 positions processed
Pass 22 complete; 0 positions processed
C>
```

See, it's a little more interesting this time, right?

Now is a good time to introduce the "-i" (information) option. Once you've got a bunch of .htb files sitting around, and you can't remember which XML control file was used to generate which tablebase, there's no need to panic. Everything from the original XML configuration is saved into the resulting tablebase, along with a bunch more information, and all of it can be retrieved from the tablebase using -i:

```
C> hoffman -i kqk.htb
Hoffman $Revision: 1.11 $ $Locker: baccala $
0 piece Nalimov tablebases found
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<tablebase offset="0x0fe4">
   <index type="compact" symmetry="8-way"/>
   <format><dtm bits="8"/></format>
   <piece color="white" type="king"/>
   <piece color="black" type="king"/>
   <piece color="white" type="queen"/>
   <futurebase filename="kk.htb"/>
   <tablebase-statistics>
      <indices>59136</indices>
      <PNTM-mated-positions>10152</PNTM-mated-positions>
      <legal-positions>47136</legal-positions>
      <stalemate-positions>115</stalemate-positions>
      <white-wins-positions>44183</white-wins-positions>
      <black-wins-positions>0</black-wins-positions>
      <forward-moves>686465</forward-moves>
      <futuremoves>2838</futuremoves>
      <max-dtm>11</max-dtm>
      <min-dtm>-11</min-dtm>
   </tablebase-statistics>
   <generation-statistics>
```

```
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Created by genctlfile.pl -->

<tablebase>
    <piece color="white" type="king"/>
    <piece color="black" type="king"/>
    <piece color="white" type="pawn"/>
    <futurebase filename="kk.htb"/>
    <futurebase filename="kqk.htb"/>
    <futurebase filename="krk.htb"/>
    <futurebase filename="kbk.htb"/>
    <futurebase filename="knk.htb"/>
    <output filename ="kpk.htb"/>
</tablebase>
```

Figure 3: `kpk.xml`

```
    <host>debian.freesoft.org</host>
    <program>Hoffman $Revision: 1.11 $ $Locker: baccala $</program>
    <args>./hoffman -g -o kqk.htb kqk.xml </args>
    <start-time>Sat Dec 16 02:11:28 2006 EST</start-time>
    <completion-time>Sat Dec 16 02:11:36 2006 EST</completion-time>
    <user-time>2.782s</user-time>
    <system-time>0.018s</system-time>
    <real-time>8.052s</real-time>

    ... about 60 more lines deleted ...

   </generation-statistics>
</tablebase>
```

We see the configuration information from the input XML file, plus a lot more information added by the program. We see the encoding scheme used to number the board positions (the `index` element), the type of information stored for each position (the `dtm` element – Distance To Mate), as well as two entire new sections — `tablebase-statistics` and `generation-statistics`. The first reports various interesting information the program determined about the tablebase, such as how many total indices there are, how many correspond to legal chess positions, how many white mates, or black mates, or stalemates there are, etc. The second reports information about the actual generation of this tablebase, like when it occured, which version of the program was used, and which computer actually computed it.

OK, so what's next? After `kk.xml` and `kqk.xml`, then you can easily understand `krk.xml`, `kbk.xml` and `knk.xml`. Once all five of these are processed, you're now ready to build `kpk.xml` (Figure 3).

Notice we've added a new type of `futurebase` — pawn promotion. Hoffman has to know what happens after that pawn transforms into a queen, rook, bishop, or knight to be able to understand what happens to the pawn!

It's starting to get more complex, right? So how do I know there isn't a bug in all of this complexity? Well, my most important blunder check is to verify the program's operation against the Nalimov tablebases. You can do this, too. If you download the appropriate Nalimov tablebases from the Internet (in this case, the two KPK files), you can verify that Hoffman's results are identical to Nalimov's using the "`-v`" (verify) and "`-n` *directory*" (location of Nalimov files) options, like this:

```
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Created by genctlfile.pl -->

<tablebase>
    <piece color="white" type="king"/>
    <piece color="black" type="king"/>
    <piece color="white" type="queen"/>
    <piece color="black" type="queen"/>
    <futurebase filename="kqk.htb" colors="invert"/>
    <futurebase filename="kqk.htb"/>
    <output filename ="kqkq.htb"/>
</tablebase>
```

Figure 4: `kqkq.xml`

```
C> ./hoffman -v -n Nalimov/ kpk.htb
Hoffman $Revision: 1.11 $ $Locker: baccala $
5 piece Nalimov tablebases found
Loading 'kpk.htb'
Verifying tablebase against Nalimov
C>
```

There were no complaints, so that means everything verified OK.

So now you've got all of the three piece tablebases. Ready to try a four piece? Figure 4 is `kqkq.xml`.

Notice several things. First, we no longer specify `kk.htb` as a futurebase. We're only interested in the single captures that lead out of the tablebase we're building. `kk.htb` has already been used to build `kqk.htb`, so its data is in there. Two queens can't be taken on a single move, so all we need to worry about is what happens if one of them is captured. That's why I use the `kqk.htb` tablebase. Notice that I use it twice, depending on which queen is being captured. The `kqk.htb` tablebase has a white queen in it, and the `colors="invert"` option to the `futurebase` element handles the case where the white queen is captured and we're left with a black queen on the board.

You'll notice also that a four piece tablebase takes a good bit longer to compute than a three piece one.

Oh, and I suppose having generating all of these tablebases, you now want to query them, huh?

You do that using the probe (-p) option, followed by a list of tablebases. Since we've got a small collection of simple tablebases, it's easiest to just load them all, like this:

```
C> ./hoffman -p *.htb
4 piece Nalimov tablebases found
Loading 'kk.htb'
Loading 'knk.htb'
Loading 'knkn.htb'
Loading 'knkp.htb'
Loading 'kpk.htb'
Loading 'kpkp.htb'
Loading 'kpkq.htb'
Loading 'kqk.htb'
```

```
Loading 'kqkn.htb'
Loading 'kqkp.htb'
Loading 'kqkq.htb'
Loading 'kqkr.htb'
Loading 'krk.htb'
Loading 'krkn.htb'
Loading 'krkp.htb'
Loading 'krkr.htb'
FEN? 8/8/8/8/p7/8/1P4k1/2K5 b
FEN 8/8/8/8/p7/8/1P4k1/2K5 b - -
Index 12658437
Draw

Nalimov score: DRAW
   g2h2    White moves and wins in 22
   g2f2    White moves and wins in 25
   g2g3    White moves and wins in 25
   g2g1    White moves and wins in 25
   g2f3    Draw
   g2h3    White moves and wins in 22
   g2f1    White moves and wins in 25
   g2h1    White moves and wins in 21
   a4a3    White moves and wins in 16
FEN or move? g2f3
FEN 8/8/8/8/p7/5k2/1P6/2K5 w - -
Index 12659332
Draw

Nalimov score: DRAW
   c1d1    Draw
   c1b1    Draw
   c1c2    Draw
   c1d2    Draw
   b2b3    Draw
   b2b4    Draw
FEN or move? b2b4
FEN 8/8/8/8/pP6/5k2/8/2K5 b - b3
Index 12593797
Draw

Nalimov score: DRAW
   f3g3    White moves and wins in 15
   f3e3    White moves and wins in 15
   f3f4    White moves and wins in 23
   f3f2    White moves and wins in 15
   f3e4    Draw
   f3g4    White moves and wins in 15
   f3e2    White moves and wins in 15
   f3g2    White moves and wins in 15
   a4xb3   Draw
FEN or move?
```
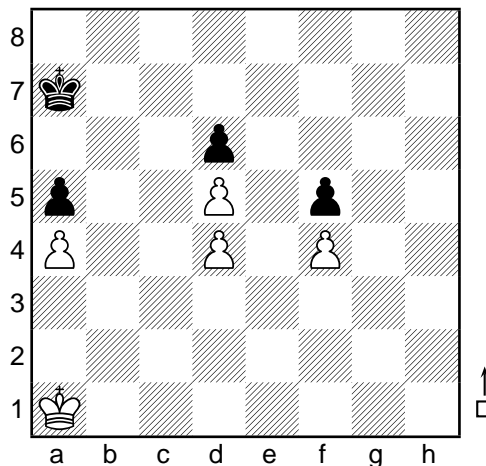
```
baccala@debian ~/src/endgame$
```

At the "FEN?" prompt you want to enter a chess position in FEN notation (you can leave off the castling rights and en passant square if you want). The program spits back its evaluation of the position (if it has one), along with a list of moves and how they evaluate. It has a history feature, so once you've typed a FEN position in once, if you end the program with a CNTL-D and not a CNTL-C, it will save everything to a history file, and you can retrieve it again on a later run using the up arrow key.

After you've put a FEN position in, you get a "FEN or move?" prompt, which allows you to enter moves and thus step forward in the game. The move parser isn't very smart right now; you can make illegal moves pretty easily and there's no way to back up. The quirkiest thing at the moment is that if you want to promote, you need to specify EXACTLY the piece you're promoting into; "b7b8=Q" is radically different from "b7b8=q"!!

The `xml` directory contains all of the XML configuration files needed to generate a complete set of three- and four-piece tablebases. These files were created by `genctlfile.pl`, a Perl script that can also create control files for five- and six-piece tablebases, though actually generating them can be quite demanding of a computer. `genalltb` is a shell script that will run Hoffman repeatedly on these files, and in the correct order, though it may take several hours to run. Hoffman has also been used to generate a complete set of five-piece tablebases, but this takes considerably longer.

Hoffman can thus duplicate much of the functionality of the Nalimov programs, but that's not all it can do. It can also duplicate much of the functionality of Eiko Bleicher's Freezer (`http://www.freezerchess.com/`). Aside from Freezer's nice GUI, the only thing Hoffman lacks is the ability to use Nalimov tablebases as futurebases, which matters because 6 piece Nalimov tablebases are available on-line, while only 5 piece tablebases are currently available for Hoffman. For example, here is a 1901 composition by Lasker and Reichhelm, used as a demo on the Freezer website:

The Hoffman XML configuration for this problem (Figure 5), is in the `xml` directory as `lasker1901.xml`. Several new features in the configuration should be apparent.

First, note the new `location` parameter that can be specified for a piece to nail it down to a particular spot. You can actually specify a list of multiple squares, such as "a4 a5 a6 a7"; we'll see an example of this later.

Note also the disappearance of the `futurebase` element; it's been replaced by a pair of `prune` statements. The program pretty much has to have either `futurebase` or `prune` statements in order to figure out how to handle things like captures. In this case, we don't use futurebases at all (so this is a stand-alone analysis), and instead tell Hoffman to regard any pawn capture as an immediate victory for the capturing side (that's the `concede` part).

Finally, let me point out the `prune-enable` elements at the beginning of the `tablebase` section. They don't do

```
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">

<tablebase>
    <prune-enable color="white" type="concede"/>
    <prune-enable color="black" type="concede"/>

    <piece color="white" type="king"/>
    <piece color="black" type="king"/>
    <piece color="white" type="pawn" location="a4"/>
    <piece color="white" type="pawn" location="d4"/>
    <piece color="white" type="pawn" location="f4"/>
    <piece color="white" type="pawn" location="d5"/>
    <piece color="black" type="pawn" location="a5"/>
    <piece color="black" type="pawn" location="f5"/>
    <piece color="black" type="pawn" location="d6"/>

    <prune color="white" move="KxP" type="concede"/>
    <prune color="black" move="KxP" type="concede"/>

    <generation-controls>
        <output filename="lasker1901.htb"/>
    </generation-controls>
</tablebase>
```

Figure 5: `lasker1901.xml`

very much, but they're very important because the program will refuse to process this tablebase without them. They're there because it's easy to lose track of pruning statements if they get buried inside futurebases. You could easily use a pruned futurebase to build a tablebase that wouldn't be accurate, but that also wouldn't be apparent just from looking at the tablebase configuration. That's why the `prune-enable` elements are there. The program will refuse to allow any pruning statement, or pruned futurebase, unless there is a matching `prune-enable` at the beginning of the `tablebase` section. This prevents "hidden" prunes from slipping into an analysis by mistake. As their name suggests, the `prune-enable` elements "turn on" pruning; the program won't prune unless they're there.

OK, so by now we know how to generate `lasker1901.htb` (`hoffman -g lasker1901.xml`, right?), and we can probe the resulting tablebase for the original problem position:

```
C> ./hoffman -p lasker1901.htb
Hoffman $Revision: 1.678 $ $Locker:  $
Loading 'lasker1901.htb'
FEN? 8/k7/3p4/p2P1p2/P2P1P2/8/8/K7 w
FEN 8/k7/3p4/p2P1p2/P2P1P2/8/8/K7 w - -
Index 177642 (lasker1901.htb)
White wins in 14
    Ka1b1     White wins in 14
    Ka1a2     Draw
    Ka1b2     Draw
8 | . . . . . . . .
7 | k . . . . . . .
6 | . . . p . . . .
5 | p . . P . p . .
```

```
4 | P . . P . P . .
3 | . . . . . . . .
2 | . . . . . . . .
1 | K . . . . . . .
  +----------------
    a b c d e f g h
WHITE to move
FEN or move?
```
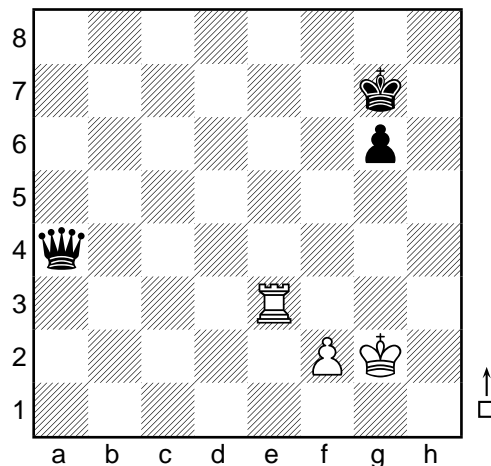
So what's the point? Can White win in 14 moves? No. White can *capture a black pawn* in 14 moves, but only if he plays Kb1! Any other move would allow Black to "draw", i.e, to prevent White from capturing a pawn without allowing Black to capture first. In fact, those other two moves allow Black to prevent White from capturing at all, but it's important to note that this Hoffman analysis doesn't guarantee that, since Black might be able to "win" by capturing a pawn. White might be able to capture a pawn, just not before Black can.

Question: What changes to the configuration would allow us to prove that Black can, in fact, completely prevent White from making any capture after a move like Ka2?

Answer: Change the black `prune-enable` and `prune` elements from `concede` to `discard`. Now the analysis will not consider any black captures, and the only kind of "draw" will be one that completely prevents white from making any kind of capture. There is one gotcha, however. White might put black in a position where his *has* to capture a pawn, but the new analysis would regard this as a stalemate. To avoid this problem, you should also add a third pruning element: `<prune color="white" move="stalemate" type="concede"/>`

Here's another example, also from the Freezer website:

It's a fortress. We're White, and we're out to show that we can draw this position. My Hoffman analysis is in the "`fortress*.xml`" files in the `xml` directory. You need to have several of the four-piece tablebases built in order to process them. I'm not going to copy them all into this document; I'll just ask you to study them on your own while I point out some salient features.

Notice how `futurebase` and `prune` statements can be used together in a single configuration file (in case you were wondering). Notice how wildcards can be used in `prune` statements. Notice how a piece can be restricted to more than one square. Notice how comments can be put into XML configurations.

Here's a final example that neither Nalimov nor Freezer can handle. There are too many pieces in play for Nalimov

```
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Diagram 67 from Rubin Fine's "Basic Chess Endings" -->

<tablebase>
   <prune-enable color="white" type="concede"/>
   <prune-enable color="black" type="concede"/>

   <piece color="white" type="king"/>
   <piece color="white" type="pawn" location="b3+"/>
   <piece color="white" type="pawn" location="c2+"/>
   <piece color="white" type="pawn" location="d3+"/>
   <piece color="black" type="king"/>
   <piece color="black" type="pawn" location="b5+"/>
   <piece color="black" type="pawn" location="c7+"/>
   <piece color="black" type="pawn" location="d5+"/>

   <futurebase filename="kpk.htb"/>
   <futurebase filename="kpk.htb" colors="invert"/>

   <prune color="white" type="concede" move="P=?"/>
   <prune color="black" type="concede" move="P=?"/>
</tablebase>
```
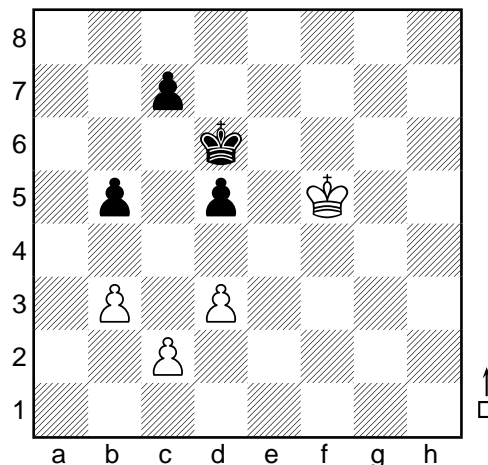
Figure 6: `fine67.xml`

(eight), and the pawn captures are too complicated for Freezer. This is position 67 from Reuben Fine's book "Basic Chess Endings":

The XML configuration (Figure 6) is `fine67.xml`, but this file can not be processed directly by Hoffman. Why? Note that pawn captures are possible from this tablebase (the plus signs in the `location` attributes mean to advance the pawn as far as it can go), yet we have not specified any `futurebase` or `prune` elements to handle those captures. Attempting to run Hoffman on this file will generate an error:

```
C> ./hoffman -o fine67.htb -g fine67.xml
Hoffman $Revision: 1.11 $ $Locker: baccala $
Malloced 40MB for tablebase entries
```

```
'kpk.htb': Too many missing pieces in futurebase
Futurebase preload failed: 'kpk.htb'
C>
```

Instead of using Hoffman directly, we need to run "pawngen" on `fine67.xml`, and it's best to do this in a directory empty except for the futurebase `kpk.htb`. Pawngen will consider all possible pawn movements, captures, and promotions, and output a series of interlinked control files that Hoffman can process. Obviously, considering all possible promotions in this case would result in positions with multiple queens that would be impossible for Hoffman to process. Pawngen, however, does honor the `prune` elements. For this example, pawngen ignored all promotion possibilities (note the `prune` elements) and created 173 control files, along with a `makefile` listing the interdependencies between them. Run the actual Hoffman analysis by typing "make". Both Perl (needed to run pawngen) and make are fairly standard on Linux and other UNIX variants, but neither are included in the Hoffman distribution. On Microsoft systems, I've used the cygwin distribution (which includes both programs) successfully.

"make" runs Hoffman, yes, 173 times and produces a directory full of tablebases. Once this is done, we use Hoffman to probe the tablebases:

```
C> ./hoffman -p *.htb
Hoffman $Revision: 1.11 $ $Locker: baccala $
0 piece Nalimov tablebases found
Loading 'Kb3b3c4Kd5.htb'
Loading 'Kb3b4c4K.htb'
Loading 'Kb3b4d3Kc7.htb'
Loading 'Kb3b4d3Kd5c7.htb'
...
Loading 'KKd6c4b5.htb'
Loading 'KKd6c4.htb'
FEN? 8/2p5/3k4/1p1p1K2/8/1P1P4/2P5/8 w
FEN 8/2p5/3k4/1p1p1K2/8/1P1P4/2P5/8 w - -
Index 12369740 (Kb3c2d3Kd5c7b5.htb)
White moves and wins in 21
Can't find Nalimov tablebase
    Pc2c3    Black moves and wins in 16
    Pc2c4    Black moves and wins in 19
    Pb3b4    White wins in 21
    Pd3d4    Draw
    Kf5g5    Black moves and wins in 13
    Kf5f6    Draw
    Kf5f4    Draw
    Kf5g6    Black moves and wins in 13
    Kf5g4    Black moves and wins in 23
FEN or move? d3d4
FEN 8/2p5/3k4/1p1p1K2/3P4/1P6/2P5/8 b - -
Index 12369849 (Kb3c2d3Kd5c7b5.htb)
Draw
Can't find Nalimov tablebase
    Pb5b4    Draw
    Kd6c6    White moves and wins in 21
    Kd6d7    White moves and wins in 36
    Kd6e7    White moves and wins in 20
```

```xml
<?xml version="1.0"?>
<!DOCTYPE tablebase SYSTEM "http://www.freesoft.org/software/hoffman/tablebase.dtd">
<!-- Diagram 67 from Rubin Fine's "Basic Chess Endings" -->

<tablebase>
    <prune-enable color="white" type="discard"/>
    <prune-enable color="black" type="concede"/>

    <piece color="white" type="king"/>
    <piece color="white" type="pawn" location="b3+"/>
    <piece color="white" type="pawn" location="c2+"/>
    <piece color="white" type="pawn" location="d3+"/>
    <piece color="black" type="king"/>
    <piece color="black" type="pawn" location="b5+"/>
    <piece color="black" type="pawn" location="c7+"/>
    <piece color="black" type="pawn" location="d5+"/>

    <futurebase filename="kpk.htb"/>
    <futurebase filename="kpk.htb" colors="invert"/>
    <futurebase filename="kqk.htb"/>
    <futurebase filename="kqkp.htb"/>
    <futurebase filename="kqpk.htb"/>

    <prune color="white" type="discard" move="P=Q"
       pawngen-condition="grep(/Q/,@white_pieces)==1"/>
    <prune color="white" type="discard" move="P=[RNB]"/>
    <prune color="black" type="concede" move="P*=?"/>
    <prune color="black" type="concede" move="?xQ"/>
    <prune color="black" type="concede" move="?xP"
       pawngen-condition="grep(/Q/,@white_pieces)==0 and $#white_pawns==0"/>
</tablebase>
```

Figure 7: `fine67a.xml`

```
    Pc7c6     White moves and wins in 20
    Pc7c5     White moves and wins in 20
FEN or move?
```

This analysis confirms Fine's analysis that b4 is White's only winning move in this position, although the pruning in the Hoffman analysis only guarantees that White can force a queen before Black can. In particular, d4 only draws because Black has the recourse b4 which allows him to blockade the position.

Of course, we'd really like to know if White can win, not just force a queen. A more sophisticated analysis (Figure 7) is `fine67a.xml`. Notice the `pawngen-condition` attributes on two of the `prune` elements. These attributes specify Perl expressions that will be evaluated by pawngen to selectively include or exclude the `prune` elements. In this example, we want to allow a *single* white queen promotion, so we unconditionally discard all white promotions into rooks, knights, or bishops, as well as white queen promotions if a white queen is already present. Thus we'll consider positions with a single white queen, but not two. In addition to promotions, we'll also concede some other moves to black that should simplify our analysis somewhat. We never want to allow black to capture the white queen, so we concede "?xQ", and there's no reason to allow black to capture the last white pawn if white hasn't queened, so we selectively prune "?xP". The syntax of this last statement is somewhat confusing unless you know that the Perl operator $\#$ counts starting at zero, so `$#white_pawns==0` actually tests if there is a single white pawn present.

Pawngen, when run on `fine67a.xml`, produces 229 control files, of which 78 contain queens, so this analysis takes longer to run than `fine67.xml` (half a day instead of an hour). When complete, its output tablebases can be queried for the original position:

```
C> ./hoffman -p *.htb
Hoffman $Revision: 1.11 $ $Locker: baccala $
0 piece Nalimov tablebases found
Loading 'Kb3b3c4Kd5.htb'
Loading 'Kb3b4c4K.htb'
Loading 'Kb3b4d3Kc7.htb'
...
Loading 'KQKd5d4.htb'
Loading 'KQKd6b5.htb'
Loading 'KQKd6c4.htb'
FEN? 8/2p5/3k4/1p1p1K2/8/1P1P4/2P5/8 w
FEN 8/2p5/3k4/1p1p1K2/8/1P1P4/2P5/8 w - -
Index 12369740 (Kb3c2d3Kd5c7b5.htb)
White moves and wins in 27
Can't find Nalimov tablebase
    Pc2c3    Black moves and wins in 11
    Pc2c4    Black moves and wins in 5
    Pb3b4    White wins in 27
    Pd3d4    Draw
    Kf5g5    Black moves and wins in 10
    Kf5f6    Draw
    Kf5f4    Draw
    Kf5g6    Black moves and wins in 10
    Kf5g4    Black moves and wins in 15
FEN or move?
```

Again, we show a white win with b4. This time, however, we know that this is actually a white *win* and not just a queening solution.

Pawngen currently accepts only a subset of Hoffman's XML syntax — no `location` attributes are allowed on any non-pawns, and pawns must have `location` attributes specifying a specific starting square followed by a plus sign. Despite these limitations, it is a key auxiliary tool that allows complicated pawn formations to be managed with a single control file.

See the Hoffman Reference Guide for more details on these and other Hoffman XML elements.