Generic Place Notation

GPN is a notation used to generate standard place notation, and hence method designs, in such a way that a method with a general structure can be generated for any valid stage.

It is based on overlaying repeated place patterns to build up place notation for a method.

GPN also allows you to focus on how methods are constructed/related by defining a method as being something else plus some embellishments.

For example, in GPN you can specify plain bob as being plain hunt plus a 12 lead end.

How to write places

In GPN you write down places in much the same way as place notation, with a few additions.

You are allowed to write `n` as place – it represents the final place in a row at the current stage. So on minor, `n` means 6.

Being able to generate methods at any stage is key to GPN, so being able to have place notation that goes to any conceivable number of bells is necessary.

GPN allows a few options here:

If you don't care about methods above stage 36, the places can notated as single characters in the standard way like this:

```
1234567890ETABCDFGHIJKLMNOPQRSUVWXYZ
           ^   ^
          12   16
```

If you want to ditch the `E` and `T` convention for eleven and twelve, you can go for the more straight-forward[1]:

```
1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
           ^   ^
          12   16
```

And finally, for handling any numbered bell at all without limit, the notation `[x]` can be used:

```
123456789[10][11][12][13][14]...
```

This `[x]` notation allows us to do a trick: we can write down 'negative' places, like this:

```
[-1]   this is the place at the end of the row (i.e. on minor, this equals 6)
[-2]   this is the place one from the end of the row (i.e. on minor, this equals 5)
[-3]   this is the place two from the end of the row (i.e. on minor, this equals 4)
[-n]   this will always be the same as 1, on any stage
```

This is useful for the genericness of GPN; for example, the halflead of reverse or double bob an even stage is always going to be `[-1][-2]`. For the odd stages, it would be `1[-2][-1]`.

There's a handy, less verbose alternative to writing something like `[-1][-2]`: if you prepend the notation with `~`, all the places are mirrored. So `~12n` is the same as `[-1][-2][-n]`.

Example: Plain Hunt

Standard PN on 4 bells is: `x.14.x.14.x.14.x.14`

Standard PN on 6 bells is: `x.16.x.16.x.16.x.16.x.16.x.16`

So the pattern we're seeing is "keep repeating the place notation pattern `x.1n` until we have `2*n` bits of place notation", with n being the stage.

In GPN we write this as a 'script' that looks like this[2]:

[1] On stages up to 16, this is the *hexadecimal* number system

[2] If this script is run through an GPN tool, it can produce the PN for any (even) stage we want

```
base="x.1n"
length="2*n"
```

Similarly, given the PN for plain hunt on odd stages:

PN on 5 bells is: `5.1.5.1.5.1.5.1.5.1`

PN on 7 bells is: `7.1.7.1.7.1.7.1.7.1.7.1.7.1.`

... the pattern is "keep repeating the place notation pattern `n.1` until we have `2*n` bits of place notation" (again, n is the stage).

In GPN we would write this as:

```
base="n.1"
length="2*n"
```

Example: Plain Bob

Plain Bob is the same as plain hunt, except that we change the last piece of place notation to `12` (on even bells).

To do this, our GPN script is as follows:

```
base="x.1n"
length="2*n"
pn[8]="12"
```

The `pn[8]="12"` line is saying "make the eighth piece of place notation be 12". This overrides the automatically generated place notation in that position taken from the `base` and `length` parts previously.

Here's a demonstration of how the `pn[...]` numbering applies to our Plain Bob Minimus place notation:

```
Plain Bob Minimus (one lead)

x      <-- pn[1]
14     <-- pn[2]
x      <-- pn[3]
14     <-- pn[4]
x      <-- pn[5]
14     <-- pn[6]
x      <-- pn[7]
12     <-- pn[8]
```

There's a problem with this GPN though: GPN is supposed to generate a method design on any number of bells, but the script given only works on 4 bells, due to the 8 in `pn[8]="12"`.

To fix this, we can rewrite that line as:

```
pn[2*n]="12"
```

There's an even easier way though: we can use negative numbers to signal we are counting the place notation position *backwards* from the end of the lead, as illustrated here[3]:

```
Plain Bob Minimus (one lead)

x      <-- pn[-8]
14     <-- pn[-7]
x      <-- pn[-6]
14     <-- pn[-5]
x      <-- pn[-4]
14     <-- pn[-3]
x      <-- pn[-2]
12     <-- pn[-1]
```

[3]This is an idea borrowed from the Python programming language

So we can specify that troublesome line as `pn[-1]="12"`.

And so now this GPN correctly generates Plain Bob on any stage of bells:

```
base="x.1n"
length="2*n"
pn[-1]="12"
```

Finally, it would be useful if we could merge the even and stage variants into one script! This can be done with the `|` operator, which specifies two expressions: one for when T is even, and one for when T is odd, in the form `<even expression>|<odd expression>`.

To illustrate, here's the script that generates Plain Bob for both even and odd stages:

```
# we use pipe operator here to mean "use x1n on even stages, and n.1 on odd stages"
base="x.1n|n.1"
length="2*n"
# we use pipe operator here to mean "use 12 on even stages, and 12n on odd stages"
pn[-1]="12|12n"
```

Finally, note the use of `#` in this script - any lines beginning with this character are 'comments' for human consumption only.

Metadata in GPN

The `name`, `info` and `author` tags are for self-explanatory stuff:

```
name="My Fantastic Method"
info="I made this method while feeding my cats"
author="Alex"
```

The metadata item `id` is special. It must be unique - so no two methods have the same `id`. Its use is for defining methods in terms of other methods.

For example, if plain hunt is defined as follows (note the the id definition):

```
id="plainhunt"
base="x.1n"
length="2*n"
```

Then we can define Plain Bob as follows:

```
id="plainbob"
base="id:plainhunt"
pn[-1]="12|12n"
```

So we have defined Plain Bob as being plain hunt plus a different bit of place notation at the lead-end.

Because the term `pn[-1]` is so useful, there's a shorthand for it: `leadend`.

So Plain Bob could also be defined as:

```
id="plainbob"
base="id:plainhunt"
leadend="12|12n"
```

Stage info

Although GPN is meant to describe methods in a general multi-stage way, sometimes there are limitations to a method, such as it being even or odd bell only, or only working on certain specific range of stages.

Such limitations can be put into the GPN script; see the following examples:

```
# this method is ony valid on odd stages
validstages=odd

# this method is only valid on even stages from 6 upwards (e.g. Cambridge Surprise)
```

```
validstages=even
minimumstage=6

# this method is only valid on even stages from 6 to 10
validstages=even
minimumstage=6
maximumstage=10

# this method is only valid on exactly these stages
validstages=7,9,11
```

Treble-bob hunting

This is the 'grid' or basis for surprise methods:

```
name="Treble Bob Hunting"
info="This is false, obviously"
validstages=even
base="x.1n.x.x"
length="4*n"
```

Further GPN examples

Reverse Bob

```
id="reverse_bob"
minimumstage=4
base="id:plain_hunt"
halflead="[-1]n|1[-1]n"
```

The last line is similar to the `leadend` shorthand: it stands for `pn[n]`.

The `[-1]n|1[-1]n` is using the usual odd-and-even versions of the place notation format.

Let's taken the latter part of that (for odd stages): `1[-1]n`. The `[-1]` part demonstrates how to refer to places counted from the end of the change: it's just shorthand for `n-1`. So on 6 bells, the whole `1[-1]n` part would translate into `156`. On 8 bells, it would translate into `178`.

And another way to define Reverse Bob, using a function:

```
id="reverse_bob"
minimumstage=4
base="id:plain_hunt"
applyfunc="reverse"
```

The `applyfunc` applies a function to the calculated place notation: it shifts the PN a half-lead, then mirrors the positions.

Example:

The place notation for PB4:

```
1234
2143  x
2413  14
4231  x
4321  14
3412  x
3142  14
1324  x
1342  12
```

So the PN shifted a half lead is:

```
x
14
x
12
x
14
x
14
```

And then we reverse the positions of that:

```
x     # x when reversed (on an even stage) is just x
14    # 14 when reversed (on stage 4) is 14 again
x
34    # 12 when rever2sed (on stage 4) is 34
x
14
x
14
```

And that is PN for reverse bob.

Double Bob

There's a few ways you could define Double Bob.

Defined as changes to plain hunt:

```
id="double_bob"
minimumstage=4
base="id:plain_hunt"
# AH: halflead would be nice as "~12|~12n" - the ~ means reverse positions places
halflead="[-1]n|1[-1]n"
leadend="12|12n"
```

The `halflead` is a shorthand similar to `leadend`: it's a stand-in for `pn[n]`.

The `[-1]` demonstrates how to refer to places counted from the end of the change: it's just shorthand for n-1.

Examples: on 6 bells, `[-1]n` is translated into `56`. On 8 bells, `[-3]n` is translated into `58`.

Here's Double Bob defined as changes to Plain Bob:

```
id="double_bob"
minimumstage=4
base="id:pain_bob"
halflead="[-1]n|1[-1]n"
```

Grandsire

```
minimumstage=5
base="x.1n|n.1"
length="2*n"
pn[1]="3n|3"
```

Little Bob

```
validstages=even
minimumstage=6
base="x.1n"
# a lead in LB is 8 changes on every stage
length="8"
```

```
pn[4]="14"
pn[-1]="12"
```

Odd Little Bob

Just for hoots; the nearest thing structurally to Little Bob on odd numbers:

```
validstages=odd
minimumstage=7
base="n.1"
length="8"
pn[4]="147"
pn[-1]="127"
```

I use the word 'pseudo' because this method has a different structure to Little Bob - it's a two-part differential, with half of the bells doing only the 'quick' work (dodges in 3-4, no 2nds), and half doing only the 'slow' work (no dodge in 3-4, make 2nds over treble). Every working bell makes 4 blows when lying (and no dodging at the back).

m-Bob

m-Bob is the generalisation of Plain Bob and Grandsire: it is the method with `m` hunt bells. What we call Plain Bob is 1-Bob, and Grandsire is 2-Bob. And if you ring Plain Bob and called a Bob at every lead, that would be the same as 3-Bob without calls in it (since at a Bob in PB, three bells end up plain hunting at the lead end).

m-Bob is an example of GPN where we need to take a parameter: the number of hunt bells `m`.

Here's the definition of m-Bob when m is an even number:

```
id="even_m_bob"
minimumstage=4
params="m"
base="x.1n|n.1"
length="2*n"
leadend="1m|1mn"
```

Here's the definition of m-Bob when m is an odd number:

```
id="odd_m_bob"
minimumstage=5
params="m"
base="x.1n|n.1"
length="2*n"
p[1]="mn|m"
```

Then we can tie the two together in one definition using the `|` operator:

```
id="m_bob"
base="id:even_n_bob|id:odd_n_bob"
```

Hints on working out the GPN for a method

Concentrate on the places, not on dodges, in much the same way as place notation; once the places are nailed down, the correct dodging, points etc just happens.

Examining the higher stages of the method is most useful. For example, looking at Bristol 8 and 10 almost gives you the pattern; but looking at 8, 10 and 16 is better.

Start from the basic 'grid' for your kind of method (e.g. plain hunt or treble bob) then add the refinements that make it into the target method.

---

alex.hunsley@gmail.com