Inventory Booking

A Proposal for an Improvement on Command-Line Bookkeeping

Martin Blais, June 2014

http://furius.ca/beancount/doc/proposal-booking

- Motivation
- Problem Description
  - Lot Date
  - Average Cost Basis
  - Capital Gains Sans Commissions
  - Cost Basis Adjustments
  - Dealing with Stock Splits
- Previous Solutions
  - Shortcomings in Ledger
  - Shortcomings in Beancount
- Requirements
  - Debugging Tools
  - Explicit vs. Implicit Booking Reduction
- Design Proposal
  - Inventory
  - Input Syntax & Filtering
  - Algorithm
  - Implicit Booking Methods
  - Dates Inserted by Default
  - Matching with No Information
  - Reducing Multiple Lots
- Examples
  - No Conflict
  - Explicit Selection By Cost
  - Explicit Selection By Date
  - Explicit Selection By Label
  - Explicit Selection By Combination
  - Not Enough Units
  - Redundant Selection of Same Lot
  - Automatic Price Extrapolation
  - Average Cost Booking

Future Work
Implementation Note
Conclusion

Motivation

The problem of "inventory booking," that is, selecting which of an inventory's trade lots to reduce when closing a position, is a tricky one. So far, in the command-line accounting community, relatively little progress has been made in supporting the flexibility to deal with many common real-life cases. This document offers a discussion of the current state of affairs, describes the common use cases we would like to be able to solve, identifies a set of requirements for a better booking method, and proposes a syntax and implementation design to support these requirements, along with clearly defined semantics for the booking method.

Problem Description

The problem under consideration is the problem of deciding, for a double-entry transaction that intends to reduce the size of a position at a particular point in time in a particular account, which of the account inventory lots contained at that point should be reduced. This should be specified using a simple data entry syntax that minimizes the burden on the user.

For example, one could enter a position in HOOL stock by buying two lots of it at different points in time:

```
2014-02-01 * "First trade"
  Assets:Investments:Stock        10 HOOL {500 USD}
  Assets:Investments:Cash
  Expenses:Commissions          9.95 USD

2014-02-15 * "Second trade"
  Assets:Investments:Stock         8 HOOL {510 USD}
  Assets:Investments:Cash
  Expenses:Commissions          9.95 USD
```

We will call the two sets of shares "trade lots" and assume that the underlying system that counts them is keeping track of each of their cost and date of acquisition separately.

The question here is, if we were to sell some of this stock, *which of the shares should we select?* Those from the first trade, or those from the second? This is an important decision, because it has an impact on capital gains, and thus on taxes. (The account of capital gains is described in more detail in the "Stock Trading" section of the Beancount cookbook if you need an explanation of this.) Depending on our financial situation, this year's trading history, and the unrealized gains that a trade would realize, sometimes we may want to select one lot over another. By now, most discount brokers even let you select your specific lot when you place a trade that reduces or closes a position.

It is important to emphasize here the two directions of inventory booking:

1. **Augmenting a position.** This is the process of creating a new trading lot, or adding to an existing trading lot (if the cost and other attributes are identical). This is easy, and amounts to adding a record to some sort of mapping that describes an account's balance (I call this an "inventory").
2. **Reducing a position.** This is generally where booking complications take place, and the problem is essentially to figure out which lot of an existing position to remove units from.

Most of this document is dedicated to the second direction.

Lot Date

Even if the cost of each lot is identical, the acquisition date of the position you're intending to close matters, because different taxation rules may apply, for example, in the US, positions held for more than 12 months are subject to a significantly lower tax rate ("the long-term capital gains rate"). For example, if you entered your position like this:

```
2012-05-01 * "First trade"
  Assets:Investments:Stock        10 HOOL {300 USD}
  Assets:Investments:Cash
  Expenses:Commissions          9.95 USD

2014-02-15 * "Second trade at same price"
  Assets:Investments:Stock         8 HOOL {300 USD}
  Assets:Investments:Cash
  Expenses:Commissions          9.95 USD
```

and we are booking a trade for 2014-03-01, selecting the first lot would result in a long-term (low) gain, because two years have passed since acquisition (position held from 2012-05-01 to 2014-03-01) while booking against the second would result in a short-term (high) capital gains tax. So it's not just a matter of the cost of the lots. Our systems don't aim to file taxes automatically at this point but we would like to enter our data in a way that eventually allows this kind of reporting to take place.

Note that this discussion assumes that you were able to *decide* which of the lots you could trade against. There are several taxation rules that may apply, and in some cases, depending on which country you live in, you may not have a choice, the government may dictate how you are meant to report your gains.

In some cases you may even need to be able to apply multiple booking methods to the same account (e.g., if an account is subject to a taxation claim from multiple countries). We will ignore this exotic case in this document, as it is quite rare.

Average Cost Basis

Things get more complicated for tax-sheltered accounts. Because there are no tax consequences to closing positions in these accounts, brokers will normally choose to account for the book value of your positions as if they were a single lot. That is, the cost of each share is computed using the weighted average of the cost of the position you are holding. This can equivalently be calculated by summing the total cost of each position and then dividing by the total number of shares. To continue with our first example:

10 HOOL x 500 USD/HOOL = 5000 USD

8 HOOL x 510 USD/HOOL = 4080 USD

Cost basis = 5000 USD + 4080 USD = 9080 USD

Total number of shares = 10 HOOL + 8 HOOL = 18 HOOL

Cost basis per share = 9080 USD / 18 HOOL = 504.44 USD/HOOL

So if you were to close some of your position and sell some shares, you would do so at a cost of 504.44 USD/HOOL. The gain you would realize would be relative to this cost; for example if you sold 5 shares at 520 USD/HOOL, your gain would be calculated like this:

(520 USD/HOOL - 504.44 USD/HOOL) x 5 HOOL = 77.78 USD

This type of booking method is made evident by two kinds of financial events that you might witness occur in these types of accounts:

1. Fees may be withdrawn by selling some shares reported at their current market price. You will see these if you hold a retirement account at Vanguard in the USA. The broker takes a fixed amount of dollars per quarter (5$, with my employer's plan) that is backed out in terms of a small, fractional number of shares of each fund to sell. That transaction is oblivious to capital gains: they simply figure out how many shares to sell to cover their fee and don't report a gain to you. You have to assume it doesn't matter. Most people don't track their gains in non-taxable accounts but freaks like us may want to compute the return nonetheless.
2. Cost basis readjustments may occur spontaneously. This is rare, but I have seen this once or twice in the case of mutual funds in a Canadian tax-deferred account: based on some undisclosed details of their trading activity, the fund management has had to issue an adjustment to the cost basis, which you are meant to apply to your positions. You get an email telling you how much the adjustment was for. 99% of people probably don't blink, don't understand and ignore this message… perhaps rightly so, as it has no impact on their taxes, but if you want to account for your trading returns in that account, you do need to count it. So we need to be able to add or remove to the cost basis of existing positions.

Note that the average cost of your positions changes on every trade and needs to get recalculated every time that you add to an existing position. A problem with this is that if you track the cost per share, these multiple recalculations may result in some errors if you store the amounts using a fixed

numerical representation (typically decimal numbers). An alternative method to track the cost in our data structures, one that is more stable numerically, is to simply keep track of the total cost instead of the cost-per-share. Also note that using fractional numbers does not provide a sufficient answer to this problem: in practice, the broker may report a specific cost, one that is rounded to specific number of decimals, and we may want to be able to book our reducing trade using that reported number rather than maintain the idealized amount that a fractional representation would. Both using a fixed decimal or a fractional representation pose problems. We will largely ignore those problems here.

In summary: we need to be able to support book against lots at their average cost, and we need to be able to adjust the cost basis of an existing position. I've been thinking about a solution to implement this that would not force an account to be marked with a special state. I think we can do this using only inventory manipulations. All we need is a small change to the syntax that allows the user to indicate to the system that it should book against the average cost of all positions.

Using the first example above, acquiring the shares would take the same form as previously, that is after buying two lots of 10 and 8 HOOL units, we end up with an inventory that holds two lots, one of 10 HOOL {500 USD} and one of 8 HOOL {510 USD}. However, when it's time to sell, the following syntax would be used (this is an old idea I've been meaning to implement for a while):

```
2014-02-01 * "Selling 5 shares at market price 550 USD"
  Assets:Investments:Stock               -5 HOOL {*}
  Assets:Investments:Cash           2759.95 USD
  Expenses:Commissions                 9.95 USD
  Income:Investments:CapitalGains
```

When the "*" is encountered in lieu of the cost, like this, it would:

1. Merge all the lots together and recalculate the average price per share (504.44 USD)
2. Book against this merged inventory, reducing the resulting lot.

After the transaction, we would end up with a single position of 13 HOOL {504.44 USD). Adding to this position at a different price would create a new lot, and those would get merged again the next time a reduction occurs at average cost. We do not need to merge lots until there is a reduction. Apart from concerns of accumulating rounding error, this solution is correct mathematically, and maintains the property that accounts aren't coerced into any specific booking method—a mix of methods could be used on every reduction. This is a nice property, even if not strictly necessary, and even if we want to be able to just specify a "default" booking method to use per account and just stick with it throughout. It's always nice to have the flexibility to support exceptions, because in real life, they sometimes occur.

Capital Gains Sans Commissions

We would like to be able to support the automatic calculation of capital gains without the commissions. This problem is described in much detail in the Commissions section of the trading documentation and in a thread on the mailing-list. The essence of the complication that occurs is that one cannot simply subtract the commissions incurred during the reporting period from the

gains that include commissions, because the commissions that were incurred to acquire the position must be pro-rated to the shares of the position that are sold. The simplest and most common way to implement this is to include the costs of acquiring the position into the cost basis of the position itself, and deduct the selling costs from the market value when a position is reduced.

Whatever new design we come up with must allow us to count these adjusted gains as this is essential to various individuals' situations. In Beancount, I have figured out a solution for this problem, which luckily enough involves only an automated transformation of a transaction flagged by the presence of a special flag on its postings... if and only if I can find a way to specify which lot to book against without having to specify its cost, because once the cost of commissions gets folded into the cost of the position, the adjusted cost does not show up anywhere in the input file, the user would have to calculate it manually, which is unreasonable. In the solution I'm proposing, the following transactions would be transformed automatically, triggered by the presence of the "C" flag:

```
2014-02-10 * "Buy"
  Assets:US:Invest:Cash    -5009.95 USD
  C Expenses:Commissions       9.95 USD
  Assets:US:Invest:HOOL       10.00 HOOL {500 USD / aa2ba9695cc7}

2014-04-10 * "Sell #1"
  Assets:US:Invest:HOOL       -4.00 HOOL {aa2ba9695cc7}
  C Expenses:Commissions       9.95 USD
  Assets:US:Invest:Cash     2110.05 USD
  Income:US:Invest:Gains

2014-05-10 * "Sell #2"
  Assets:US:Invest:HOOL       -6.00 HOOL {aa2ba9695cc7}
  C Expenses:Commissions       9.95 USD
  Assets:US:Invest:Cash     3230.05 USD
  Income:US:Invest:Gains
```

They would be automatically transformed by a plugin into the following and replace the original transactions above:

```
2014-02-10 * "Buy"
  Assets:US:Invest:Cash        -5009.95 USD
  X Expenses:Commissions           9.95 USD
  X Income:US:Invest:Rebates      -9.95 USD
  X Assets:US:Invest:HOOL         10.00 HOOL {500.995 USD / aa2ba9695cc7}

2014-04-10 * "Sell #1"
  X Assets:US:Invest:HOOL         -4.00 HOOL {aa2ba9695cc7}
  X Expenses:Commissions           9.95 USD
  X Income:US:Invest:Rebates      -9.95 USD
  Assets:US:Invest:Cash         2110.05 USD
  Income:US:Invest:Gains ; Should be (530-500)*4 - 9.95*(4/10) - 9.95 =
                         ; 106.07 USD
```

```
2014-05-10 * "Sell #2"
  X Assets:US:Invest:HOOL         -6.00 HOOL {aa2ba9695cc7}
  X Expenses:Commissions           9.95 USD
  X Income:US:Invest:Rebates      -9.95 USD
  Assets:US:Invest:Cash         3230.05 USD
  Income:US:Invest:Gains ; Should be (540-500)*6 - 9.95*(6/10) - 9.95 =
                         ; 224.08 USD
```

The "X" flag would just mark the postings that have been transformed, so maybe they can be rendered with a special color or attribute later on. The rebates account is included as a separate counter just to make the transaction balance.

The reason I need to relax the disambiguation of trading lots is that the user needs to be able to specify the matching leg without having to specify the cost of the position, because at the point where the position is reduced (2014-04-10), there is no way to figure out what the cost of the original lot was. Just to be clear, in the example above, this means that if all the information have is 4 HOOL and 500 USD, there is no way to back out a cost of 500.995 USD that could specify a match with the opening trade lot, because that happened with 10 HOOLs.

So we need to be more explicit about booking. In the example above, I'm selecting the matching lot "by label," that is, the user has the option to provide a unique lot identifier in the cost specification, and that can later on be used to disambiguate which lot we want to book a reduction/sale against. The example above uses the string "`aa2ba9695cc7`" in this way.

(An alternative solution to this problem would involve keep track of *both* the original cost (without commissions), and the actual cost (with commissions), and then finding the lot against the former, but using the latter to balance the transaction. This idea is to allow the user to keep using the cost of a position to select the lot, but I'm not even sure if that is possible, in the presence of various changes to the inventory. More thought is required on this matter.)

Cost Basis Adjustments

In order to adjust the cost basis, one can replace the contents of an account explicitly like this:

```
2014-03-15 * "Adjust cost basis from 500 USD to 510 USD"
  Assets:US:Invest:HOOL      -10.00 HOOL {500 USD}
  Assets:US:Invest:HOOL       10.00 HOOL {510 USD}
  Income:US:Invest:Gains
```

This method works well and lets the system automatically compute the gains. But it would be nice to support making adjustments in price units against the total cost of the position, for example, "add 340.51 USD to the cost basis of this position". The problem is that the adjusted cost per share now needs to be computed by the user… it's inconvenient. Here's one way we could support this well, however:

```
2014-03-15 * "Adjust cost basis from 500 USD to 510 USD"
  Assets:US:Invest:HOOL      -10.00 HOOL {500 USD}
```

```
  Assets:US:Invest:HOOL       10.00 HOOL {}
  Income:US:Invest:Gains    -340.51 USD
```

If all the legs are fully specified - they have a calculatable balance - we allow a single price to be elided. This solves this problem well.

Dealing with Stock Splits

Accounting for stock splits creates some complications in terms of booking. In general, the problem is that we need to deal with changes in the meaning of a commodity over time. For example, you could do this in Beancount right now and it works:

```
2014-01-04 * "Buy some HOOL"
  Assets:Investments:Stock        10 HOOL {1000.00 USD}
  Assets:Investments:Cash

2014-04-17 * "2:1 split into Class A and Class B shares"
  Assets:Investments:Stock       -10 HOOL {1000.00 USD}
  Assets:Investments:Stock        10 HOOL  {500.00 USD}
  Assets:Investments:Stock        10 HOOLL {500.00 USD}
```

One problem with splitting lots this way is that the price and meaning of a HOOL unit before and after the split differs, but let's just assume we're okay with that for now (this can be solved by *relabeling* the commodity by using another name, ideally in a way that is not visible to the user - we have a pending discussion elsewhere).

The issue that concerns booking is, when you sell that position, which cost do you use? The inventory will contain positions at 500 USD, so that's what you should be using, but is it obvious to the user? We can assume this can be learned with a recipe .

Now, what if we automatically attach the transaction date? Does it get reset on the split and now you would have to use 2014-04-17? If so, we could not automatically inspect the list of trades to determine whether this is a long-term vs. short-term trade. We need to somehow preserve some of the attributes of the original event that augmented this position, which includes the original trade date (not the split date) and the original user-specified label on the position, if one was provided.

Forcing a Single Method per Account

For accounts that will use the average booking method, it may be useful to allow specifying that an account should only use this booking method. The idea is to avoid data entry errors when we know that an account is only able to use this method.

One way to ensure this is to automatically aggregate inventory lots when augmenting a position. This ensures that at any point in time, there is only a single lot for each commodity. If we associated an inventory booking type to each account, we could define a special one that also triggers aggregation upon the addition of a position.

Another approach would be to not enforce these aggregations, but to provide a plugin that would check that for those accounts that are marked as using the average cost inventory booking method by default, that only such bookings actually take place.

Previous Solutions

This section reviews existing implementations of booking methods in command-line bookkeeping systems and highlights specific shortcomings that motivate why we need to define a new method.

Shortcomings in Ledger

(Please note: I'm not deeply familiar with the finer details of the inner workings of Ledger; I inferred its behavior from building test examples and reading the docs. If I got any of this wrong, please do let me know by leaving a comment.)

Ledger's approach to booking is quite liberal. Internally, Ledger does not distinguish between conversions held at cost and regular price conversions: all conversions are assumed held at cost, and the only place trading lots appear is at reporting time (using its `--lots` option).

Its "`{}`" cost syntax is meant to be used to disambiguate lots on a position reduction, not to be used when acquiring a position. The cost of acquiring a position is specified using the "`@`" price notation:

```
2014/05/01 * Buy some stock (Ledger syntax)
  Assets:Investments:Stock        10 HOOL @ 500 USD
  Assets:Investments:Cash
```

This will result in an inventory of 10 HOOL {500 USD}, that is, the cost is *always* stored in the inventory that holds the position:

```
$ ledger -f l1.lgr bal --lots
10 HOOL {USD500} [2014/05/01]
            USD-5000  Assets:Investments
            USD-5000    Cash
10 HOOL {USD500} [2014/05/01]    Stock
--------------------
10 HOOL {USD500} [2014/05/01]
            USD-5000
```

There is no distinction between conversions at cost and without cost—all conversions are tracked as if "at cost." As we will see in the next section, this behaviour is distinct from Beancount's semantics, which requires a conversion held at cost to be specified using the "`{}`" notation for both its augmenting and reducing postings, and distinguishes between positions held at cost and price conversions.

The advantage of the Ledger method is a simpler input mechanism, but it leads to confusing outcomes if you accumulate many conversions of many types of currencies in the same account. An inventory can easily become fragmented in its lot composition. Consider what happens, for instance,

if you convert in various directions between 5 different currencies… you may easily end up with USD held in CAD, JPY, EUR and GBP and vice-versa… any possible combinations of those are possible. For instance, the following currency conversions will maintain their original cost against their converted currencies:

```
2014/05/01 Transfer from Canada
  Assets:CA:Checking             -500 CAD
  Assets:US:Checking              400 USD @ 1.25 CAD

2014/05/15 Transfers from Europe
  Assets:UK:Checking             -200 GBP
  Assets:US:Checking              500 USD @ 0.4 GBP

2014/05/21 Transfers from Europe
  Assets:DE:Checking             -100 EUR
  Assets:US:Checking              133.33 USD @ 0.75 EUR
```

This results in the following inventory in the `Assets:US:Checking` account:

```
$ ledger -f l2.lgr bal --lots US:Checking
500.00 USD {0.4 GBP} [2014/05/15]
133.33 USD {0.75 EUR} [2014/05/21]
400.00 USD {1.25 CAD} [2014/05/01]  Assets:US:Checking
```

The currencies are treated like stock. But nobody thinks of their currency balances like this, one normally thinks of currencies held in an account as units in and of themselves, not in terms of their price related to some other currency. In my view, this is confusing. After these conversions, I just think of the balance of that account as 1033.33 USD.

A possible rebuttal to this observation is that most of the time a user does not care about inventory lots for accounts with just currencies, so they just happen not print them out. Out of sight, out of mind. But there is no way to distinguish when the rendering routine should render lots or not. If instructed to produce a report, a balance routine should ideally only render lots only for some accounts (e.g., investing accounts, where the cost of units matters) and not for others (e.g. accounts with currencies that were deposited sometimes as a result of conversions). Moreover, when I render a balance sheet, for units held at cost we need to report the book values rather than the number of shares. But for currencies, the currency units themselves need to get reported, always. If you don't distinguish between the two cases, how do you know which to render? I think the answer is to select which view you want to render using the options. The problem is that neither provides a single unified view that does the correct thing for both types of commodities. The user should not have to specify options to view partially incorrect views in one style or the other, this should be automatic and depend on whether we care about the cost of those units. This gets handled correctly if you distinguish between the two kinds of conversions.

But perhaps most importantly, this method does not particularly address the *conversion problem*: it is still possible to create money out of thin air by making conversions back and forth at different rates:

```
2014/05/01 Transfer to Canada
  Assets:US:Checking             -40000 USD
  Assets:CA:Checking              50000 CAD @ 0.80 USD

2014/05/15 Transfer from Canada
  Assets:CA:Checking             -50000 CAD @ 0.85 USD
  Assets:US:Checking              42500 USD
```

This results in a trial balance with just 2500 USD:

```
$ ledger -f l3.lgr bal
             2500 USD  Assets:US:Checking
```

This is not a desirable outcome. We should require that the trial balance always sum to zero (except for virtual transactions). After all, this is the aggregate realization of the elegance of the double-entry method: because all transactions sum to zero, the total sum of any subgroup of transactions also sums to zero… except it doesn't. In my view, any balance sheet that gets rendered should comply with the accounting equation, precisely.

In my explorations with Ledger, I was hoping that by virtue of its inventory tracking method it would somehow naturally deal with these conversions automatically and thus provide a good justification for keeping track of all units at cost, but I witness the same problem showing up. (The conversions issue has been a real bitch of a problem to figure out a solution for in Beancount, but it is working, and I think that the same solution could be applied to Ledger to adjust these sums, without changing its current booking method: automatically insert a special conversion entry at strategic points - when a balance sheet is rendered.) In the end, I have not found that tracking costs for every transaction would appear to have an advantage over forgetting the costs for price conversions, the conversions problem is independent of this. I'd love to hear more support in favour of this design choice.

Now, because all conversions maintain their cost, Ledger needs to be quite liberal about booking, because it would be inconvenient to force the user to specify the original rate of conversion for each currency, in the case of commonly occurring currency conversions. This has an unfortunate impact on those conversions which we do want to hold at cost: arbitrary lot reductions are allowed, that is, reductions against lots that do not exist. For example, the following trade does not trigger an error in Ledger:

```
2014/05/01 Enter a position
  Assets:Investments:Stock        10 HOOL @ 500 USD
  Assets:Investments:Cash

2014/05/15 Sale of an impossible lot
  Assets:Investments:Stock       -10 HOOL {505 USD} @ 510 USD
  Assets:Investments:Cash
```

This results in an inventory with a long position of 10 HOOL and a short position of 10 HOOL (at a different cost):

```
$ ledger -f l4.lgr bal --lots stock
10 HOOL {USD500} [2014/05/01]
   -10 HOOL {USD505}  Assets:Investments:Stock
```

I think that the reduction of a 505 USD position of HOOL should not have been allowed; Beancount treats this as an error by choice.

Ledger clearly appears to support booking against an existing inventory, so surely the intention was to be able to reduce against existing positions. The following transactions do result in an empty inventory, for instance:

```
2014/05/01 Enter a position
  Assets:Investments:Stock      10 HOOL @ 500 USD
  Assets:Investments:Cash

2014/05/15 Sale of matching lot
  Assets:Investments:Stock     -10 HOOL {500 USD} [2014/05/01]  @ 510 USD
  Assets:Investments:Cash
```

With output:

```
$ ledger -f l5.lgr bal --lots
              USD100  Equity:Capital Gains
```

As you see, the HOOL position has been eliminated. (Don't mind the auto-inserted equity entry in the output, I believe this will be fixed in a pending patch.) However, both the date and the cost must be specified for this to work. The following transactions results in a similar problematic long + short position inventory as mentioned above:

```
2014/05/01 Enter a position
  Assets:Investments:Stock      10 HOOL @ 500 USD
  Assets:Investments:Cash

2014/05/15 Sale of matching lot?
  Assets:Investments:Stock     -10 HOOL {500 USD} @ 510 USD
  Assets:Investments:Cash
```

With output:

```
$ ledger -f l6.lgr bal --lots
10 HOOL {USD500} [2014/05/01]
   -10 HOOL {USD500}  Assets:Investments:Stock
              USD100  Equity:Capital Gains
--------------------
10 HOOL {USD500} [2014/05/01]
   -10 HOOL {USD500}
              USD100
```

This is a bit surprising, I expected the lots to book against each other. I suspect this may be an unreported bug, and not intended behaviour.

Finally, the "{}" cost syntax is allowed be used on augmenting legs as well. The documentation points to these methods being equivalent. It results in an inventory lot that does not have a date associated with it, but the other leg is not converted to cost:

```
2014/05/01 Enter a position
  Assets:Investments:Stock      10 HOOL {500 USD}
  Assets:Investments:Cash
```

With output:

```
$ ledger -f l7.lgr bal --lots
                   0  Assets:Investments
   -10 HOOL {USD500}    Cash
    10 HOOL {USD500}    Stock
--------------------
                   0
```

I probably just don't understand how this is meant to be used; in Beancount the automatically computed leg would have posted a -5000 USD amount to the Cash account, not shares.

What I think is going on, is that Ledger (probably) accumulates all the lots without attempting to match them together to try to make reductions, and then at reporting time - and only then - it matches the lots together based on some reporting options:

- Group lots by both cost/price and date, using --lots
- Group lots by cost/price only, using --lot-prices
- Group lots by date only, using --lot-dates

It is not entirely obvious from the documentation how that works, but the behavior is consistent with this.

(If this is correct, I believe that to be a problem with its design: the mechanism by which an inventory reduction is booked to an existing set of lots should definitely not be a reporting feature. It needs to occur before processing reports, so that a single and definitive history of inventory bookings can be realized. If variants in bookings are supported - and I don't think this is a good idea - they should be supported before the reporting phase. In my view, altering inventory booking strategies from command-line options is a bad idea, the strategies in place should be fully defined by the language itself.)

Shortcomings in Beancount

Beancount also has various shortcomings, though different ones.

In contrast to Ledger, Beancount disambiguates between currency conversions and conversions "held at cost":

```
2014-05-01 * "Convert some Loonies to Franklins"
  Assets:CA:Checking          -6000 CAD
  Assets:Investments:Cash      5000 USD @ 1.2 CAD   ; Currency conversion
```

```
2014-05-01 * "Buy some stock"
  Assets:Investments:Stock        10 HOOL {500 USD} ; Held at cost
  Assets:Investments:Cash
```

In the first transaction, the `Assets:Investment:Cash` account results in an inventory of 5000 USD, with no cost associated to it. However, after the second transaction the `Assets:Investment:Stock` account has an inventory of 10 HOOL with a cost of 500 USD associated to it. This is perhaps a little bit more complex, and requires more knowledge from the user: there are two kinds of conversions and he has to understand and distinguish between these two cases, and this is not obvious for newcomers to the system. Some user education is required (I'll admit it *would* help if I wrote more documentation).

Beancount's method is also less liberal, it requires a strict application of lot reduction. That is, if you enter a transaction that reduces a lot that does not exist, it will output an error. The motivation for this is to make it difficult for the user to make a mistake in data entry. Any reduction of a position in an inventory has to match against exactly one lot.

There are a few downsides that result from this choice:

- It requires the users to always find the cost of the lot to match against. This means that automating the import of transactions requires manual intervention from the user, as he has to go search in the ledger to insert the matching lot. (Note that theoretically the import code could load up the ledger contents to list its holdings, and if unambiguous, could look for the cost itself and insert it in the output. But this makes the import code dependent on the user's ledger, which most import codes aren't doing). This is an important step, as finding the correct cost is required in order to correctly compute capital gains, but a more flexible method is desired, one that allows the user to be a bit more lazy and not have to put the cost of the existing position when the choice is obvious, e.g., when there is a single lot of that unit to match against. I'd like to relax this method somewhat.

- The strict requirement to reduce against an existing lot also means that both long and short positions of the same commodities "held at cost" in the same account cannot exist. This is not much of a problem in practice, however, because short positions are quite rare—few individuals engage in them—and the user could always create separate accounts to hold short positions if needed. In fact, it is already customary to create a dedicated subaccount for each commodity in an investment account, as it naturally organizes trading activity nicely (reports a bit of a mess otherwise, it's nice to see the total position value grouped by stock, since they offer exposure to different market characteristics). Different accounts should work well, as long as we treat the signs correctly in reductions, i.e., buying stock against an existing short position is seen as a position reduction (the signs are just reversed).

Another problem with the Beancount booking method is in how it matches against lots with dates. For example, if you were to try to disambiguate between two lots at the same price, the most obvious input would not work:

```
2012-05-01 * "First trade"
  Assets:Investments:Stock        10 HOOL {500 USD}
  Assets:Investments:Cash

2014-02-15 * "Second trade at very same price"
  Assets:Investments:Stock         8 HOOL {500 USD}
  Assets:Investments:Cash

2014-06-30 * "Trying to sell"
  Assets:Investments:Stock        -5 HOOL {500 USD / 2012-05-01}
  Assets:Investments:Cash
```

This would report a booking error. This is confusing. In this case Beancount, in its desire to be strict, applies strict matching against the inventory lots, and attempting to match units of a lot of (HOOL, 500 USD, 2012-05-01) with units of (HOOL, 500 USD, *None*) simply fails. Note that the lot-date syntax is accepted by Beancount on both augmenting and reducing legs, so the "solution" is that the user is forced to specifically provide the lot-date on acquiring the position. This would work, for instance:

```
2012-05-01 * "First trade"
  Assets:Investments:Stock        10 HOOL {500 USD / 2012-05-01}
  Assets:Investments:Cash

2014-02-15 * "Second trade at very same price"
  Assets:Investments:Stock         8 HOOL {500 USD}
  Assets:Investments:Cash

2014-06-30 * "Trying to sell"
  Assets:Investments:Stock        -5 HOOL {500 USD / 2012-05-01}
  Assets:Investments:Cash
```

The inconvenience here is that when the user entered the first trade, he could not predict that he would enter another trade at the same price in the future and typically would not have inserted the lot-date. More likely than not, the user had to go back and revisit that transaction in order to disambiguate this. We can do better.

This is a shortcoming of its implementation, which reflects the fact that position reduction was initially regarded as an exact matching problem rather than a fuzzy matching one—it wasn't clear to me how to handle this safely at the time. This needs to be fixed after this proposal, and what I'm doing with this document is very much a process of clarifying for myself what I want this fuzzy matching to mean, and to ensure that I come up with an unambiguous design that is easy to enter data for. Ledger always automatically attaches the transaction date to the inventory lot, and in my view this is the correct thing to do (below we propose attaching more information as well).

Context

[Updated on Nov'2014]

It is important to distinguish between two types of booking:

1. **Booking.** The strict type of booking that occurs when we are considering the entire list of transactions. We will call this the "booking" process, and it should occur just once. This process should be strict: the failure of correctly book a reduction to an existing lot should trigger a fatal error.
2. **Inventory Aggregation.** When we are considering a subset of entries, it is possible that entries that add some lots are removed, and that other entries that reduce or remove those lots are kept. This creates a situation in which it is necessary to support aggregations over time of changes to an inventory that may not admit a matching lot. If we are to support arbitrary subsets of transactions being aggregated, we must take this into account. This aggregation process should never lead to an error.

Let's take an example to justify why we need the non-booking summarization. Let's say we have a stock purchase and sale:

```
2013-11-03 * "Buy"
  Assets:Investments:VEA      10 AAPL {37.45 USD} ;; (A)
  Assets:Investments:Cash     -370.45 USD

2014-08-09 * "Sell"
  Assets:Investments:VEA     -5 AAPL {37.45 USD}
  Assets:Investments:Cash     194.40 USD
  Income:Investments:PnL
```

If we filter "by year" and only want to view transactions that occurred in 2014, AND we don't "close" the previous year, that is, we do not create opening balances entries that would deposit the lots in the account at the beginning of the year - this could happen if we filter by tag, or by some other combination of conditions - then the (A) lot is not present in the inventory to be debited from. We must somehow accommodate a -5 AAPL at that point. When reporting, the user could decide how to summarize and "match up as best it can" those legs, but converting them to units or cost, or convert them to a single lot at average-cost.

The booking process, however, should admit no such laxity. It should be strict and trigger an error if a lot cannot be matched. This would only be applied on the total set of transactions, and only once, never on any subset. The purpose of the booking stage should be threefold:

1. **Match** against existing lots and issue errors when that is impossible. Do this, using the method specified by the user.
2. **Replace** all partially specified lots by a full lot specification, the lot that was matched during the booking process.

3. **Insert links** on transactions that form a trade: a buy and corresponding sells should be linked together. This essentially identifies trades, and can then be used for reporting later on. (Trades at average cost will require special consideration.)

We can view Ledger's method as having an aggregation method only, lacking a booking stage. The particular method for aggregation is chosen using --lots or --lot-dates or --lot-prices. This would indicate that a "booking" stage could be tacked on to Ledger without changing much of its general structure: it could be inserted as a pre- or post- process, but it would require Ledger to sort the transactions by date, something it does not currently do (it runs a single pass over its data).

Beancount has had both methods for a while, but the separate nature of these two operations has not been explicitly stated so far—instead, the inventory supported an optional flag to raise an error when booking was impossible. Also, inventory aggregation has not been thought as requiring much customization, it was meant to be unified to the booking process. It is now clear that they are two distinct algorithms, and that a few different methods for aggregating can be relevant (though they do not function the same as Ledger's do. Note: you can view the inventory aggregation method as a GROUP BY within an Inventory object's lots: which columns do you group over? This requires use cases). In any case, fully specified lots should match against each other by default: we need to support this as the degenerate case to use for simple currencies (not held at cost)... we would not want to accumulate all changes in the same currency as separate lots in an inventory, they need to cross each other as soon as they are applied.

We want to change this in order to make the code clearer: there should be a separate "booking" stage, provided by a **plugin**, which resolves the partially specific lot reduction using the algorithm outlined in this document, and a separate method for inventory aggregation should not bother with any of those details. In fact, the inventory aggregation could potentially simply become an accumulated list of lots, and the summarization of them could take place a posteriori, with some conceptual similarly to when Ledger applies its aggregation. Just using a simple aggregation becomes more relevant once we begin entering more specific data about lots, such as always having an acquisition date, a label, and possibly even a link to the entry that created the lot. In order to trigger summarization sensibly, functions to convert and summarize accumulated inventories could be provided in the shell, such as "UNITS(inventory)".

Precision for Interpolation

The interpolation capabilities will be extended to cover eliding a single number, any number, in any subset of postings whose "weights" resolve to a particular currency (e.g., "all postings with weights in USD"). The interpolation needs to occur at a particular precision. The interpolated number should be quantized automatically, and the number of fractional digits to quantize it to should be automatically inferred from the DisplayContext which is itself derived from the input file. Either the most common or the maximum number of digits witnessed in the file should be used.

For a use case, see this thread:

> *On Mon, Nov 24, 2014 at 12:33 PM, ELI <eliptus@gmail.com> wrote:*

Regarding the last piece on which there seems to be a misunderstanding between you and I, let me provide an standalone example, outside the context of the plugin. Since Vanguard only makes three decimal places of precision available me, I need to have the actual share count calculated from the price and transaction amount.

For example, I have a transaction on my activities page with these details:
Shares Transacted: 0.000
Share Price: 48.62
Amount: 0.02

The only way to enter this transaction and have it balance would be to manually calculate the Shares Transacted with four decimal places of precision. My preferred way of entering this would be to enter the Share Price and Amount and have Beancount calculate Shares Transacted to a precision associated with my Vanguard account. Additionally, as metadata, I'd record "Shares Transacted: 0.000" as history of "what the statement said".

Maybe you can give me your thoughts on how such a case would be addressed with planned Beancount features or as a plugin I could right?

What does the downloadable file contain? 0.000 or 0.0004 or even 0.00041?

What this is, most likely, is the quarterly fee that they take, which they price in terms of shares. If this is your Vanguard account, and you sum all the similar such transactions around it, you should observe that it sums to 5$ per quarter.

I would log it like this:

```
2014-11-23 * "Quarterly fee"
  Assets:US:Vanguard:VIIPX      -0.00041 VIIPX {* 48.62 USD}
  Expenses:Financial:Fees    0.02 USD
```

The "*" will be required to merge all the lots into a single one (average cost booking) because the 48.62 USD they declare is _not_ one of your lots, but rather just the price of the asset that happened to be there on the day they took the fee. In other words, they take their fee in terms of units of each asset, priced at the current price, deducting against the cost basis of all your lots together (it doesn't matter which because this is a pre-tax account, so no capital gains are reported).

Now you're asking, how could I avoid calculating 0.00041 myself?

My first answer would be: let's first see if the OFX download from Vanguard includes the precision. That would solve it. I believe it does (I checked my own history of downloads and I have some numbers in there with 4 fractional digits).

My second answer would be that - if you don't have the number of shares from the file - after I implement the much needed new inventory booking proposal (http://furius.ca/beancount/doc/proposal-booking), the interpolation capabilities will be vastly extended, such that eliding the number of shares would even be possible, like this:

```
2014-11-23 * "Quarterly fee"
  Assets:US:Vanguard:VIIPX      VIIPX {* 48.62 USD}
  Expenses:Financial:Fees    0.02 USD
```

Now this by itself still would not solve the problem: this would store 0.000411353353 which is limited at 12 fractional digits because of the default context. So that's incorrect. **What would need to be done to deal with this is to infer the most common number of digits used on units of VIIPX and to quantize the result to that number of digits (I think this could be safe enough). The number of digits appearing in the file for each currency** is already tracked in a DisplayContext object that is stored in the options_map from the parser. I'll have to take that into account in the inventory booking proposal. I'm adding this to the proposal.

Requirements

The previous sections introduce the general problem of booking, and point out important shortcomings in both the Beancount and Ledger implementations. This section will present a set of desires for a new and improved booking mechanism for command-line bookkeeping software implementations.

Here are reasonable requirements for a new method:

- **Explicit inventory lot selection** needs to be supported. The user should be able to indicate precisely which of the lots to reduce a position against, per-transaction. An account or commodity should not have to have a consistent method applied across all its trades.
- Explicit inventory lot selection should **support partial matching**, that is, in cases where the lots to match against are ambiguous, the user should be able to supply only minimal information to disambiguate the lots, e.g., the date, or a label, or the cost, or combinations of these. For instance, if only a single lot is available to match against, the user should be able to specify the cost as "`{}`", telling the system to book at cost, but essentially saying: "book against any lot." This should trigger an error only if there are multiple lots.
- A variety of informations should be supported to specify an existing lot, and they should be combinable:
    - The cost of acquisition of the position;
    - The date the position was acquired at;
    - A user-provided label.
- Cases where insufficient information is provided for explicit lot selection is available should admit for a **default booking method** to be invoked. The user should be able to specify **globally** and **per-account** what the default booking method should be. This paves the way for **implicit lot selection**. A degenerate method should be available that kicks off an error and requires the user to be explicit.
- **Average cost booking** needs to be supported, as it is quite common in retirement or tax-sheltered accounts. This is the default method used in Canadian investment accounts.

- **Cost basis adjustments** need to be supported as well. The problem should be able to be specified by providing a specific lot (or all of a commodity's lots at average cost) and a *dollar amount* to adjust the position by.
- **Stock splits** need to be able to **maintain some of the original attributes of the position**, specifically, the original trade date and the user-specified label, if one has been provided. This should allow a common syntax to specify an original trading lot when reducing a position after a split.
- Reducing a position needs to **always book against an existing lot**. I'm preserving the Beancount behavior here, which I've argued for previously, as it works quite well and is a great method to avoid data entry mistakes. This naturally defines an invariant for inventories: *all positions of a particular commodity held at cost should be of the same sign in one inventory* (this can be used as a sanity check).
- **Bookings that change the sign of a number of units should raise an error**, unless an explicit exception is requested (and I'm not even convinced that we need it). Note again that bookings only occur for units held at cost, so currency conversions are unaffected by this requirement. Beancount has had this feature for a while, and it has proved useful to detect errors. For example, if an inventory has a position of 8 HOOL {500 USD} you attempt to post a change of -10 units to this lot, the resulting number of units is now negative: -2. This should indicate user error. The only use case I can think for allowing this is the trading of futures spreads and currencies, which would naturally be reported in the same account (a short position in currencies is not regarded as a different instrument in the same way that a short position would); this is the only reason to provide an exception, and I suspect that 99% of users will not need it.]

Debugging Tools

Since this is a non-trivial but greatly important part of the process of entering trading data, we should provide tools that list in detail the running balance for an inventory, including all of the detail of its lots.

Booking errors being reported should include sufficient context, that is:

- The transaction with offending posting
- The offending posting
- The current state of the inventory before the posting is applied, with all its lots
- The implicit booking method that is in effect to disambiguate between lots
- A detailed reason why the booking failed

Explicit vs. Implicit Booking Reduction

Another question that may come up in the design of a new booking method is whether we require the user to be *explicit* about whether he thinks this is an addition to a position, or a reduction. This could be done, for instance, by requiring a slightly different syntax for the cost, for example "`{}`"

would indicate an addition and "[ ]" a reduction. I'm not convinced that it is necessary to make that distinction, maybe the extra burden on the user is not worth it, but it might be a way to cross-check an expectation against the actual calculations that occur. I'll drop the idea for now.

Design Proposal

This section presents a concrete description of a design that fulfills the previously introduced requirements. We hope to keep this as simple as possible.

Inventory

An inventory is simply a list of lot descriptors. Each inventory lot will be defined by:

    UNITS, (CURRENCY, COST-UNITS, COST-CURRENCY, DATE, LABEL)

Fields:

- UNITS: the number of units of that lot that are held
- CURRENCY: the type of commodity held, a string
- COST-UNITS: the number in the price of that lot
- COST-CURRENCY: the pricing commodity of the cost
- DATE: the acquisition date of the lot
- LABEL: an arbitrary user-specified string used to identify this lot

If this represents a lot held at cost, after this processing, only the LABEL field should be optionally with a NULL value. Anyone writing code against this inventory could expect that all other values are filled in with non-NULL values (or are otherwise all NULL).

Input Syntax & Filtering

The input syntax should allow the specification of cost as any combination of the following informations:

- The **cost per unit**, as an amount, such as "`500 USD`"
- A **total cost**, to be automatically divided by the number of units, like this: `{... +9.95 USD}`. You should be able to use either cost per unit, total cost, or even combine the two, like this: `{500 + 9.95 USD}`. This is useful to enter commissions. This syntax also replaces the previous {{...}} syntax.
- The **lot-date**, as a YYYY-MM-DD date, such as "`2014-06-20`"
- A **label**, which is any non-numerical identifier, such as first-apple or some random uuid like "`aa2ba9695cc7`"
- A **special marker** "*" that indicates to book at the average cost of the inventory balance

These following postings should all be valid syntax:

```
...
  Assets:Investments:Stock        10 HOOL {500 USD}      ; cost
  Assets:Investments:Stock        10 HOOL {339999615d7a} ; label
  Assets:Investments:Stock        10 HOOL {2014-05-01}   ; lot-date
  Assets:Investments:Stock        10 HOOL {}             ; no information

... Combinations
  Assets:Investments:Stock        10 HOOL {500 USD, 2014-05-01}
  Assets:Investments:Stock        10 HOOL {2014-05-01, 339999615d7a}
```

Algorithm

All postings should be processed in a separate step after parsing, in the order of their date, against a running inventory balance for each account. The cost amount should become fully determined at this stage, and if we fail to resolve a cost, an error should be raised and the transaction deleted from the flow of directives (after the program loudly complaining about it).

When processing an entry, we should match and filter all inventory lots against all the filters that are provided by the user. In general, after filtering the lots with the user restrictions:

- If the set of lots is empty, an error should be raised;
- If there is a single lot, this lot should be selected (success case);
- If there are multiple lots, the **default implicitly booking method** for the corresponding account should be invoked.

Implicit Booking Methods

If there are multiple matching lots to choose from during booking, the following implicit booking methods could be invoked:

- **`STRICT`**: Select the only lot of the inventory. If there are more than one lots, raise an error.
- **`FIFO`**: Select the lot with the earliest date.
- **`LIFO`**: Select the lot with the latest date.
- **`AVERAGE`**: Book this transaction at average cost.
- **`AVERAGE_ONLY`**: Like `AVERAGE` but also trigger an aggregation on an addition to a position. This can be used to enforce that the only booking method for all lots is at the average cost.
- **`NONE`**: Don't perform any inventory booking on this account. Allow a mix of lots for the same commodity or positive and negative numbers in the inventory. (This essentially devolves to the Ledger method of booking.)

This method would *only* get invoked if disambiguation between multiple lots is required, after filtering the lots against the expression provided by the user. The `STRICT` method is basically the degenerate disambiguation which issues an error if there is any ambiguity and should be the default.

There should be a default method that applies to all accounts. The default value should be overridable. In Beancount, we would add a new "option" to allow the user to change this:

```
option "booking_method" "FIFO"
```

The default value used in a particular account should be specifiable as well, because it is a common case that the method varies by account, and is fixed within the account. In Beancount, this would probably become part of the open directive, something like this:

```
2003-02-17 open Assets:Ameritrade:HOOL  HOOL  booking:FIFO
```

(I'm not sure about the syntax.)

Resolving Same Date Ambiguity

If the automatic booking method gets invoked to resolve an ambiguous lot reduction, e.g. FIFO, if there are multiple lots at the same date, something needs to be done to resolve which of the lots is to be chosen. The line number at which the transaction appears should be selected. For example, in the following case, a WIDGET Of 8 GBP would be selected:

```
2014-10-15 * "buy widgets"
  Assets:Inventory     10 WIDGET {} ;; Price inferred to 8 GBP/widget
  Assets:Cash         -80 GBP

2014-10-15 * "buy another widget"
  Assets:Inventory      1 WIDGET {} ;; Price inferred to 9 GBP/widget
  Assets:Cash          -9 GBP

2014-10-16 * "sell a widget"
  Assets:Cash           11 GBP
  Assets:Inventory      -1 WIDGET {} ;; Ambiguous lot
```

Dates Inserted by Default

By default, if an explicit date is not specified in a cost, the date of the transaction that holds the posting should be attached to the trading lot automatically. This date is overridable so that stock splits may be implemented by a transformation to a transaction like this:

```
2014-01-04 * "Buy some HOOL"
  Assets:Investments:Stock   10 HOOL {1000.00 USD}
  Assets:Investments:Cash

2014-04-17 * "2:1 split into Class A and Class B shares"
  Assets:Investments:Stock  -10 HOOL {1000.00 USD, 2014-01-04} ; reducing
  Assets:Investments:Stock   10 HOOL  {500.00 USD, 2014-01-04} ; augment
  Assets:Investments:Stock   10 HOOLL {500.00 USD, 2014-01-04} ; augment
```

Matching with No Information

Supplying no information for the cost should be supported and is sometimes useful: in the case of *augmenting* a position, if all the other legs have values specified, we should be able to automatically infer the cost of the units being traded:

```
2012-05-01 * "First trade"
  Assets:Investments:Stock            10 HOOL {}
  Assets:Investments:Cash       -5009.95 USD
  Expenses:Commissions              9.95 USD
```

In the case of *reducing* a position—and we can figure that out whether that is the case by looking at the inventory at the point of applying the transaction to its account balance—an empty specification should trigger the default booking method. If the method is STRICT, for instance, this would select a lot *only* if there is a single lot available (the choice is unambiguous), which is probably a common case if one trades infrequently. Other methods will apply as they are defined.

Note that the user still has to specify a cost of "`{}`" in order to inform us that this posting has to be considered "held at cost." This is important to disambiguate from a price conversion with no associated cost.

Reducing Multiple Lots

If a single trade needs to close multiple existing lots of an inventory, this can dealt with trivially by inserting one posting for each lot. I think this is a totally reasonable requirement. This could represent a single trade, for instance, if your brokerage allows it:

```
2012-05-01 * "Closing my position"
  Assets:Investments:Stock           -10 HOOL {500 USD}
  Assets:Investments:Stock           -12 HOOL {510 USD}
  Assets:Investments:Cash       12000.00 USD
  Income:Investments:Gains
```

Note: If the cost basis specification for the lot matches multiple lots of the inventory and the result is unambiguous, the lots will be correctly selected. For example, if the ante-inventory contains just those two lots (22 HOOL in total), you should be able to have a single reducing posting like this:

```
2012-05-01 * "Closing my position"
  Assets:Investments:Stock           -22 HOOL {}
  Assets:Investments:Cash       12000.00 USD
  Income:Investments:Gains
```

and they will both be included.

On the other hand, if the result is ambiguous (for example, if you have more than these two lots) the booking strategy for that account would be invoked. By default, this strategy will be "STRICT" which will generate an error, but if this account's strategy is set to "FIFO" (or is not set and the default global strategy is "FIFO"), the FIFO lots would be automatically selected for you, as per your wish.

Lot Basis Modification

Another idea is to support the modification of a specific lot's cost basis in a single leg. The way this could work, is by specifying the number of units to modify, and the "+ number currency" syntax would be used to adjust the cost basis by a specific number, like this:

```
2012-05-01 * "Adjust cost basis by 250 USD for 5 of the 500 USD units"
  Assets:Investments:Stock            ~5 HOOL {500 + 250 USD}
```

If the number of units is smaller than the total number of units in the lot, split out the lot before applying the cost basis adjustment. *I'm not certain I like this.*

Not specifying the size could also adjust the entire position (I'm not sure if this makes sense as it departs from the current syntax significantly):

```
2012-05-01 * "Adjust cost basis by 250 USD for the 500 USD units"
  Assets:Investments:Stock            HOOL {500 + 250 USD}
```

In any case, all the fields other than the total cost adjustment would be used to select which lot to adjust.

Tag Reuse

We will have to be careful in the inventory booking to warn on reuse of lot labels. Labels should be unique.

Examples

Nothing speaks more clearly than concrete examples. If otherwise unspecified, we are assuming that the booking method on the `Assets:Investments:Stock` account is `STRICT`.

No Conflict

Given the following inventory lots:

21, (HOOL, 500, USD, 2012-05-01, null)

22, (AAPL, 380, USD, 2012-06-01, null)

This booking should succeed:

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {}
  ...
```

This booking should fail (because the amount is not one of the valid lots):

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {520 USD}
  ...
```

This one too should fail (no currency matching lot):

```
2013-05-01 *
  Assets:Investments:Stock           -10 MSFT {80 USD}
  ...
```

So should this one (invalid date when a date is specified):

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {500 USD, 2010-01-01}
  ...
```

Explicit Selection By Cost

Given the following inventory:

21, (HOOL, 500, USD, 2012-05-01, null)

32, (HOOL, 500, USD, 2012-06-01, “abc”)

25, (HOOL, 510, USD, 2012-06-01, null)

This booking should succeed:

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {510 USD}
  ...
```

This booking should fail if the stock account’s method is STRICT (ambiguous):

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {500 USD}
  ...
```

This booking should succeed if the stock account’s method is FIFO, booking against the lot at 2012-05-01 (earliest):

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {500 USD}
  ...
```

Explicit Selection By Date

Given the same inventory as previously, this booking should succeed:

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {2012-05-01}
  ...
```

This booking should fail if the method is STRICT (ambiguous)

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {2012-06-01}
  ...
```

Explicit Selection By Label

This booking should succeed, because there is a single lot with the “abc“ label:

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {abc}
  ...
```

If multiple lots have the same label, ambiguous cases may occur; with this inventory, for example:

32, (HOOL, 500, USD, 2012-06-01, “abc”)

31, (HOOL, 510, USD, 2012-07-01, “abc”)

The same booking should fail.

Explicit Selection By Combination

With the initial inventory, this booking should succeed (unambiguous):

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {500 USD, 2012-06-01}
  ...
```

There is only one lot at a cost of 500 USD and at an acquisition date of 2012-06-01.

Not Enough Units

The following booking would be unambiguous, but would fail, because there aren’t enough units of the lot in the inventory to subtract from:

```
2013-05-01 *
  Assets:Investments:Stock           -33 HOOL {500 USD, 2012-06-01}
  …
```

Redundant Selection of Same Lot

If two separate postings select the same inventory lot in one transaction, it should be able to work:

```
2013-05-01 *
  Assets:Investments:Stock           -10 HOOL {500 USD, 2012-06-01}
  Assets:Investments:Stock           -10 HOOL {abc}
  ...
```

Of course, if there are not enough shares, it should fail:

```
2013-05-01 *
  Assets:Investments:Stock           -20 HOOL {500 USD, 2012-06-01}
  Assets:Investments:Stock           -20 HOOL {abc}
  ...
```

Automatic Price Extrapolation

If all the postings of a transaction can have their balance computed, we allow a single price to be automatically calculated - this should work:

```
2014-03-15 * "Adjust cost basis from 500 USD to 510 USD"
  Assets:US:Invest:HOOL      -10.00 HOOL {500.00 USD}
  Assets:US:Invest:HOOL       10.00 HOOL {}
  Income:US:Invest:Gains    -340.51 USD
```

And result in the equivalent of:

```
2014-03-15 * "Adjust cost basis from 500 USD to 510 USD"
  Assets:US:Invest:HOOL      -10.00 HOOL {500.00 USD}
  Assets:US:Invest:HOOL       10.00 HOOL {534.51 USD}
  Income:US:Invest:Gains    -340.51 USD
```

Of course, preserving the original date of the lot should work too, so this should work as well:

```
2014-03-15 * "Adjust cost basis from 500 USD to 510 USD"
  Assets:US:Invest:HOOL      -10.00 HOOL {500.00 USD}
  Assets:US:Invest:HOOL       10.00 HOOL {2014-02-04}
  Income:US:Invest:Gains    -340.51 USD
```

Average Cost Booking

Augmenting lots with the average cost syntax should fail:

```
2014-03-15 * "Buying at average cost, what does this mean?"
  Assets:US:Invest:Stock      10.00 HOOL {*}
  Income:US:Invest:Gains   -5000.00 USD
```

Reducing should work, this is the main use case:

```
2014-03-15 * "Buying a first lot"
  Assets:US:Invest:Stock      10.00 HOOL {500.00 USD}
  Assert:US:Invest:Cash    -5000.00 USD

2014-04-15 * "Buying a second lot"
  Assets:US:Invest:Stock      10.00 HOOL {510.00 USD}
  Assets:US:Invest:Cash    -5100.00 USD

2014-04-28 * "Obtaining a dividend in stock"
  Assets:US:Invest:Stock         1.00 HOOL {520.00 USD}
  Income:US:Invest:Dividends  -520.00 USD
```

```
2014-05-20 * "Sell some stock at average cost"
  Assets:US:Invest:Stock      -8.00 HOOL {*}
  Assets:US:Invest:Cash     4240.00 USD
  Income:US:Invest:Gains    ; Automatically calculated: -194.29 USD
```

The effect would be to aggregate the 10 HOOL {500.00 USD} lot, the 10 HOOL {510.00 USD} lot, and the 1.00 HOOL {520.00 USD} lot, together into a single lot of 21 HOOL {505.714285 USD}, and to remove 8 HOOL from this lot, which is 8 HOOL x 505.714285 USD/HOOL = 4045.71 USD. We receive 4240.00 USD, which allows us to automatically compute the capital gain: 4240.00 - 4045.71 = 194.29 USD. After the reduction, a single lot of 13 HOOL {505.714285 USD} remains.

It should also work if we have multiple currencies in the account, that is adding the following transaction to the above should not make it faile:

```
2014-04-15 * "Buying another stock"
  Assets:US:Invest:Stock      15.00 AAPL {300.00 USD}
  Assets:US:Invest:Cash    -4500.00 USD
```

However it should fail if we have the same currency priced in distinct cost currencies in the same account:

```
2014-03-15 * "Buying a first lot"
  Assets:US:Invest:Stock      10.00 HOOL {500.00 USD}
  Assert:US:Invest:Cash    -5000.00 USD

2014-04-15 * "Buying a second lot"
  Assets:US:Invest:Stock      10.00 HOOL {623.00 CAD}
  Assets:US:Invest:Cash    -62300.00 CAD

2014-05-20 * "Sell some stock at average cost"
  Assets:US:Invest:Stock      -8.00 HOOL {*}  ; Which HOOL, USD or CAD?
  Assets:US:Invest:Cash    4240.00 USD
  Income:US:Invest:Gains
```

But of course, this *never* happens in practice, so I'm not too concerned. We could potentially support specifying the cost-currency to resolve this case, that is, replacing the sell leg with this:

```
2014-05-20 * "Sell some stock at average cost"
  Assets:US:Invest:Stock      -8.00 HOOL {* USD}
  Assets:US:Invest:Cash    4240.00 USD
  Income:US:Invest:Gains
```

I'm quite sure it wouldn't be useful, but we could go the extra mile and be as general as we can possibly be.

Future Work

This proposal does not yet deal with cost basis re-adjustments! We need to way to be able to add or remove a fixed *dollar amount* to a position's cost basis, that is, from (1) a lot identifier and (2) a

cost-currency amount, we should be able to update the cost of that lot. The transaction still has to balance.

Here are some ideas what this might look like:

```
2014-05-20 * "Adjust cost basis of a specific lot"
  Assets:US:Invest:Stock       10.00 HOOL {500.00 USD + 230.00 USD}
  Income:US:Invest:Gains     -230.00 USD
```

The resulting inventory would be 10 HOOL at 523.00 USD per share, and the rest of the original lots, less 10 HOOL at 500.00 USD per share (there might be some of that remaining, that's fine). This is the equivalent of

```
2014-05-20 * "Adjust cost basis of a specific lot"
  Assets:US:Invest:Stock      -10.00 HOOL {500.00 USD}
  Assets:US:Invest:Stock       10.00 HOOL {523.00 USD}
  Income:US:Invest:Gains     -230.00 USD
```

except you did not have to carry out the calculation. This should therefore be implemented as a transformation. However, this is not super useful, because this is already supported... adjustments are usually performed on lots at average cost, and this is where the problem lie: you'd have to make the calculation yourself! Here's a relevant use case:

```
2014-05-20 * "Adjust cost basis of a specific lot"
  Assets:US:Invest:Stock       10.00 HOOL {* + 230.00 USD}
  Income:US:Invest:Gains     -230.00 USD
```

This would first average all the lots of HOOL together and recompute the cost basis with the new amount. This should work:

```
2014-03-15 * "Buying a first lot"
  Assets:US:Invest:Stock       5.00 HOOL {500.00 USD}
  Assert:US:Invest:Cash    -2500.00 USD

2014-04-15 * "Buying a second lot"
  Assets:US:Invest:Stock        5.00 HOOL {520.00 USD}
  Assets:US:Invest:Cash     -2600.00 USD

2014-05-20 * "Adjust cost basis of the Hooli units"
  Assets:US:Invest:Stock       10.00 HOOL {* + 230.00 USD}
  Income:US:Invest:Gains     -230.00 USD
```

Notice how we still have to specify an amount of HOOL shares to be repriced. I like this, but I'm wondering if we should allow specifying "all the units" like this (this would be a more radical change to the syntax):

```
2014-05-20 * "Adjust cost basis of the Hooli units"
  Assets:US:Invest:Stock           * HOOL {* + 230.00 USD}
  Income:US:Invest:Gains     -230.00 USD
```

Perhaps the best way to auto-compute cost basis adjustments would be via powerful interpolation, like this:

```
2014-05-20 * "Adjust cost basis of a specific lot"
  Assets:US:Invest:Stock      -10.00 HOOL {500 USD}  ; reduce
  Assets:US:Invest:Stock       10.00 HOOL {}         ; augment, interpolated
  Income:US:Invest:Gains     -230.00 USD
```

See the Smarter Elision document for more details on a proposal.

Inter-Account Booking

Some tax laws require a user to book according to a specific method, and this might apply between all accounts. This means that some sort of transfer needs to be applied in order to handle this correctly. See the separate document for detail.

TODO - complete this with more tests for average cost booking!

Implementation Notes

Separate Parsing & Interpolation

In order to implement a syntax for reduction that does not specify the cost, we need to make changes to the parser. Because there may be no costs on the posting, it becomes impossible to perform balance checks at parsing time. Therefore, we will need to postpone balance checks to a stage *after* parsing. This is reasonable and in a way nice: it is best if the Beancount parser does not output many complex errors at that stage. A new "post-parse" stage will be added, right before running the plugins.

Conclusion

This document is work-in-progress. I'd really love to get some feedback in order to improve the suggested method, which is why I put this down in words instead of just writing the code. I think a better semantic can be reached by iterating on this design to produce something that works better in all systems, and this can be used as documentation later on for developers to understand why it is designed this way.

Your feedback is much appreciated.