Dealing with stack allocations

Brute idea

For cases when there is a single `store` instruction to memory allocated with `alloca`, my idea was to simply drop those allocations and use the value that is to be stored directly. This would result in `%3` becoming an alias for `%0` with `%2` being fully omitted. Potential issue I could already see here is “pointer aliasing”, and thus I think this approach is not generally useful. As for loops, we could try to identify the “loop variables” by looking at which ones of them use a store instruction inside code that “loops back” in the control flow and are allocated before the loop starts. I think it will be useful to analyse the CFG of the code to help extract this information.

```
define dso_local i32 @identity(i32 noundef %0) #0 {
  %2 = alloca i32, align 4
  store i32 %0, ptr %2, align 4
  %3 = load i32, ptr %2, align 4
  ret i32 %3
}
```

This would then result in this code before translating to E-PEG. Or probably with the bitcast omitted and `ret i32 %0` instead.

```
define dso_local i32 @identity(i32 noundef %0) #0 {
  %3 = bitcast i32 %0 to i32
  ret i32 %3
}
```

Equality Saturation: Engineering Challenges and Applications

In the paper (p. 146) it is suggested to construct special E-PEG nodes to denote a stack allocation. This probably makes more sense in the general case, as it enables to do optimization that eliminate the allocations later during equality saturation. Similarily to constant folding, I think it would be preferred to handle this during saturation, not in a separate phase. Nevertheless, I'm having a bit of trouble understanding how this is supposed to work. As you can see there there are those `RHO_SIGMA` and `RHO_VALUE` nodes which are not children of the `ALLOCA` node. Can you give me some pointers as to what they could constitute?

Phi nodes

I have managed to write code that does generate phi nodes. Basically, what I found that is that if ones uses ternary operators or boolean logic, clang happily converts those to phi nodes. Below is a simple example of a program that uses phi nodes.

C code

```c
int phi(int a, int b, int c) {
    int r = (c ? a : b);
    return r;
}
```

Corresponding LLVM IR with the phi instruction.

```llvm
define dso_local i32 @phi(i32 noundef %0, i32 noundef %1, i32 noundef %2) #0 {
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  store i32 %0, ptr %4, align 4
  store i32 %1, ptr %5, align 4
  store i32 %2, ptr %6, align 4
  %8 = load i32, ptr %6, align 4
  %9 = icmp ne i32 %8, 0
  br i1 %9, label %10, label %12

10:                                               ; preds = %3
  %11 = load i32, ptr %4, align 4
  br label %14

12:                                               ; preds = %3
  %13 = load i32, ptr %5, align 4
  br label %14

14:                                               ; preds = %12, %10
  %15 = phi i32 [ %11, %10 ], [ %13, %12 ]
  store i32 %15, ptr %7, align 4
  %16 = load i32, ptr %7, align 4
  ret i32 %16
}
```

The phi nodes serve as a nice way to omit allocations completely and translate very easily into the corresponding E-Node.