Rewriting math equations into basic UREs

Based on the work of Gusev and Evans [1][2], we can translate math equations into UREs, following the simple steps below. Going through an example, we would find that this process is pretty much mechanical and intuitive, and only middle-school level of math knowledge is needed. So do not be scared by the math symbols. We strongly recommend a beginner repeats the steps of an example, during which the beginner would quickly master the skills. These skills are applicable to other real world problems, not limited to the examples here.

Use auto-regressive filter for example:

Step	Equations		Initial conditions	
1: Iterative form	$y_i = \sum_{j=1}^{n} c_j * y_{i-j} + x_i$	$i = 1, ..., n$		$y_{i-j} = 0$ if $i - j \leq 0$
2: Recursive form	$y_i = y_i + c_j * y_{i-j}$	$i, j = 1, ..., n$	$y_i = x_i$	$y_{i-j} = 0$ if $i - j \leq 0$
3: DSA	$y_i^j = y_i^{j-1} + c_j * y_{i-j}^n$	$i, j = 1, ..., n$	$y_i^0 = x_i$	$y_{i-j}^n = 0$ if $i - j \leq 0$
4: Full index form	$y_i^j = y_i^{j-1} + c_j^0 * y_{i-j}^n$	$i, j = 1, ..., n$	$y_i^0 = x_i^0$	$y_{i-j}^n = 0$ if $i - j \leq 0$
5: UREs	$C_i^j = C_{i-1}^j$ $Y_i^j = Y_{i-1}^{j-1}$ $y_i^j = y_i^{j-1} + C_i^j * Y_i^j$	$i, j = 1, ..., n$	$C_{i-1}^j = c_j^0$ $Y_{i-1}^{j-1} = y_{i-j}^n$ $y_i^0 = x_i^0$	if $i = 1$ if $i = 1$ or $j = 1$ $y_{i-j}^n = 0$ if $i - j \leq 0$

Step 1: Iterative form

First, write down the math equations of the original problem. Usually, these equations are to iterate over a domain (like $j = 1, \ldots, n$) and compute some variables (like y_i).

Step 2: Recursive form

Translate the iterative form to recursive form is straightforward: figure out the initial value of a variable (e.g. $y_i = x_i$), and update the variable every iteration with a new value based on its previous value (e.g. $y_i = y_i + \ldots$).

Step 3: DSA (Dynamic Single Assignment)

Every iteration, assign a variable to a distinct memory location. Every writing to a variable (and correspondingly, every reading of the variable) are indicated by the iteration index. For example, in iteration j, $y_i = y_i + \ldots y_{i-j}$ is changed into $y_i^j = y_i^{j-1} + \ldots y_{i-j}^n$. After such renaming, the iterations where y are assigned values are clearly exposed. Consequently, the dependences between these iterations are made explicit.

For another example, the initial condition $y_i = x_i$ is changed into $y_i^0 = x_i$: iteration index j starts from 1 in this example, and therefore, y_i^0 is the y_i before iteration j starts. In general, if iteration index j starts from s and has a step h, the initial condition should be $y_i^{s-h} = \ldots$.

Step 4: Full index form

Now variables are referenced with full indices, but constants are not: c_j and x_i are input values, never modified during the iterations. Give them full indices as well by adding 0's: c_j and x_i are changed into c_j^0 and x_i^0. After this, variables and constants will be processed in the same way.

At this point, the equations we get are AREs (Affine Recurrence Equations), that is, every data flow (write after read dependence) has a distance vector in the form of $d = Az + d_0$, where A is a matrix, z is the iteration indices $\begin{pmatrix} i \\ j \end{pmatrix}$, and d_0 is a constant vector:

Dependence No.	Write	Read	Dependence distance
1	y_i^j	y_i^{j-1}	$\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} i \\ j-1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
2	y_i^j	c_j^0	$\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} j \\ 0 \end{pmatrix} = \begin{pmatrix} i-j \\ j \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$
3	y_i^j	y_{i-j}^n	$\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} i-j \\ n \end{pmatrix} = \begin{pmatrix} j \\ j-n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ -n \end{pmatrix}$

Step 5: UREs

There are still some

Of course, depending on the domain in a specific problem, one should

This is because the three occurences of the variable y

. new value of variable y_i is assigned to a distinct memory location named y_i^j, and its reference to a previous value of variable y_i is changed to

This renaming

In the illustrated example, y_i is the result of a series of additions,

here it is

```
    // Or so
    /* or so */
    for (int i=1; i< 99; i++) {
        int j=10;
    // dkfjldjfl
    //lkdfjdsf
    }
```

And after

for1

for2