Transforming Experiences

ICSE99/STS

James M. Neighbors
Bayfront Technologies, Inc.
1280 Bison B9-231
Newport Beach, CA 92626 USA
+1 714 436 0322
James.Neighbors@BayfrontTechnologies.com

Path (1969-1984 at University of California, Irvine)

- Existing program improvement (1974 w/Tim Standish)
- Existing program specialization (1976 w/Tim Standish)
- Domains for program synthesis (1978 w/Peter Freeman)
- Using application domains (1984 w/industry)

Beliefs

1. **Every transformation has enabling conditions**. Even the most innocuous of transformations (e.g., ?A*(?B+?C) => (?A*?B+?A*?C)) have enabling conditions. Enabling conditions can help guide transformation because they can prune choices.

2. **Metalevel planning is necessary**. We saw the application of transformations form into patterns when we applied them one at a time. These patterns infer plans that drive these patterns. Applying individual transformations one at a time is tedious and error prone. Low-level tactics and high-level strategies are necessary.

3. **Source-to-source transformations work well for specialization and optimization**. It is easier to find and remove unused generalization than to generalize something that is specific. The latter requires the addition of knowledge not in the code being manipulated.

4. **Transform at the correct level of abstraction**. The “correct” level is a level where the concepts you are manipulating are directly represented. Once again do not try to infer any knowledge that's already been removed from the code.

5. **Optimization and refinement are not the same thing**. Optimization works for all implementations of the objects being manipulated. Optimization does not change the level of abstraction. Refinements make an implementation choice, change the level of abstraction, and are irreversible. Refinements remove knowledge from the code.

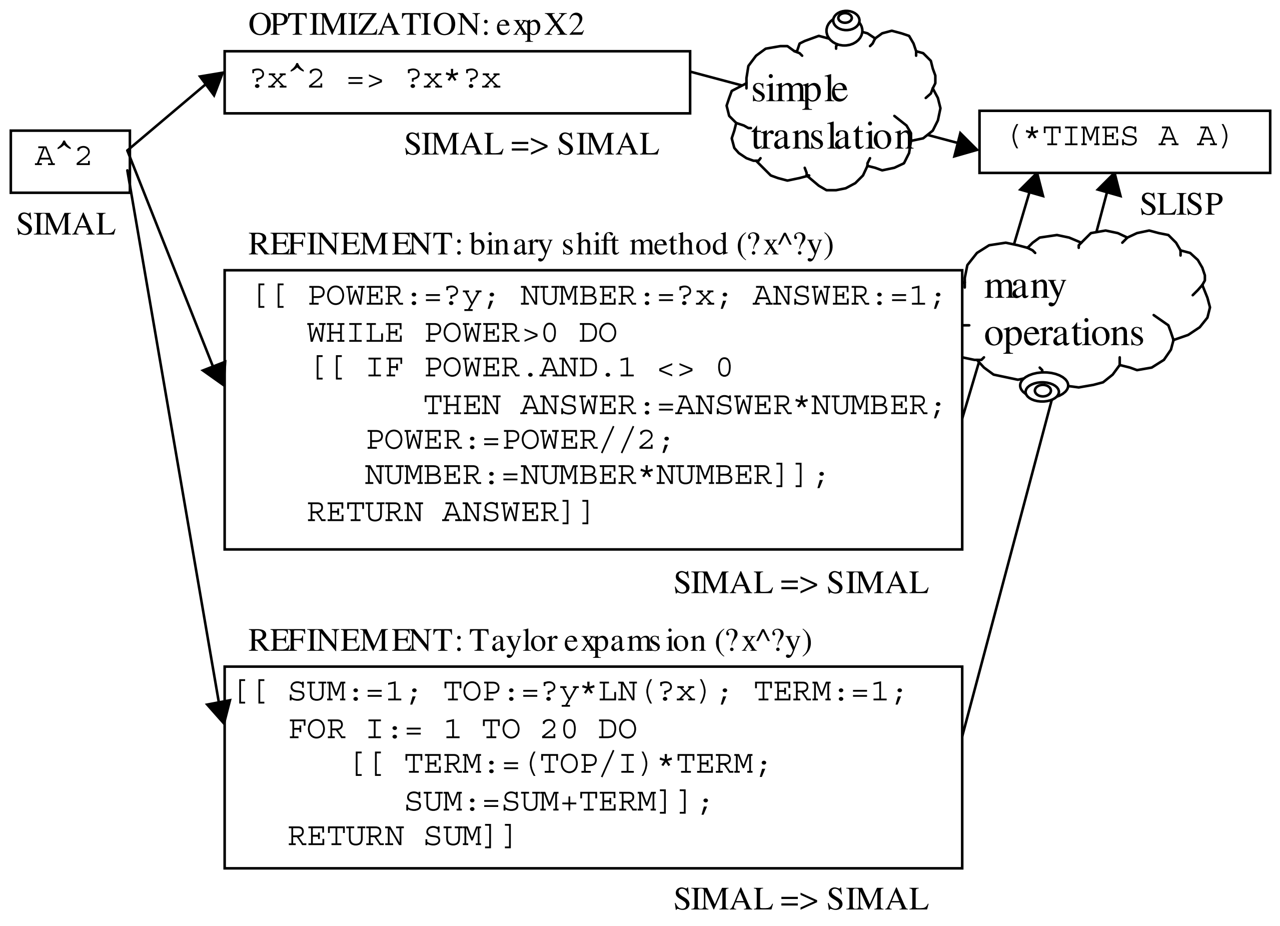
6. **Use narrow-spectrum languages rather than wide-spectrum languages**. Wide-spectrum languages include all of the formal theory languages and general specification languages. In order to achieve the “correct” level of abstraction as we have characterized it above, wide-spectrum languages will have to use abstraction notations. Once they do, they revert to narrow-spectrum languages with cumbersome notations.

7. **Reduce transformation mechanism power to achieve useful strategic planning**. If you use incredibly complex transformations, then it is hard understand what they do. Worse, it's hard to characterize what they do for strategic planning purposes.

8. **Architecture evolves as a consequence of implementation techniques in addition to system function.** There are many ways to implement the same construct. Inline code, threaded code, and threaded code interpreters are all valid schemes for implementing a single component.

9. **Synthesis is easier than understanding**. It's tempting to search for the right-hand side of a transformation and claim program understanding. However you must have enough knowledge to synthesize before you can recognize.

10. **Domains provide education**. This is the largest impact of this work. I never would have guessed it at the time.



```
?x^2 => ?x*?x
```

```
[[ POWER:=?y; NUMBER:=?x; ANSWER:=1;
   WHILE POWER>0 DO
    [[ IF POWER.AND.1 <> 0
          THEN ANSWER:=ANSWER*NUMBER;
       POWER:=POWER//2;
       NUMBER:=NUMBER*NUMBER]];
   RETURN ANSWER]]
```

```
[[ SUM:=1; TOP:=?y*LN(?x); TERM:=1;
   FOR I:= 1 TO 20 DO
      [[ TERM:=(TOP/I)*TERM;
         SUM:=SUM+TERM]];
   RETURN SUM]]
```

Figure 1. Transformation implementation of exponentiation