Metaprogram Implementation

by Second Order Source Transformation

James R. Cordy*

Medha Shukla Sarkar†

School of Computing, Queen's University

Kingston, Canada

* ITC-IRST, Trento, Italy

† Middle Tennessee State University, Nashville, U.S.A.

The ASDT Project

- Joint IBM / Queen's project started in late 80's
- Goal: Round trip engineering (design-code-design-...)
- Subgoals:
  - Design recovery (from source code)
  - Design instantiation ("metaprogramming")
- Technologies:
  - Theory-model paradigm (from mathematics)
  - Design as E-R model (Prolog facts) of E-R theory (Prolog schema)
  - Transfer technology source transformation (TXL)

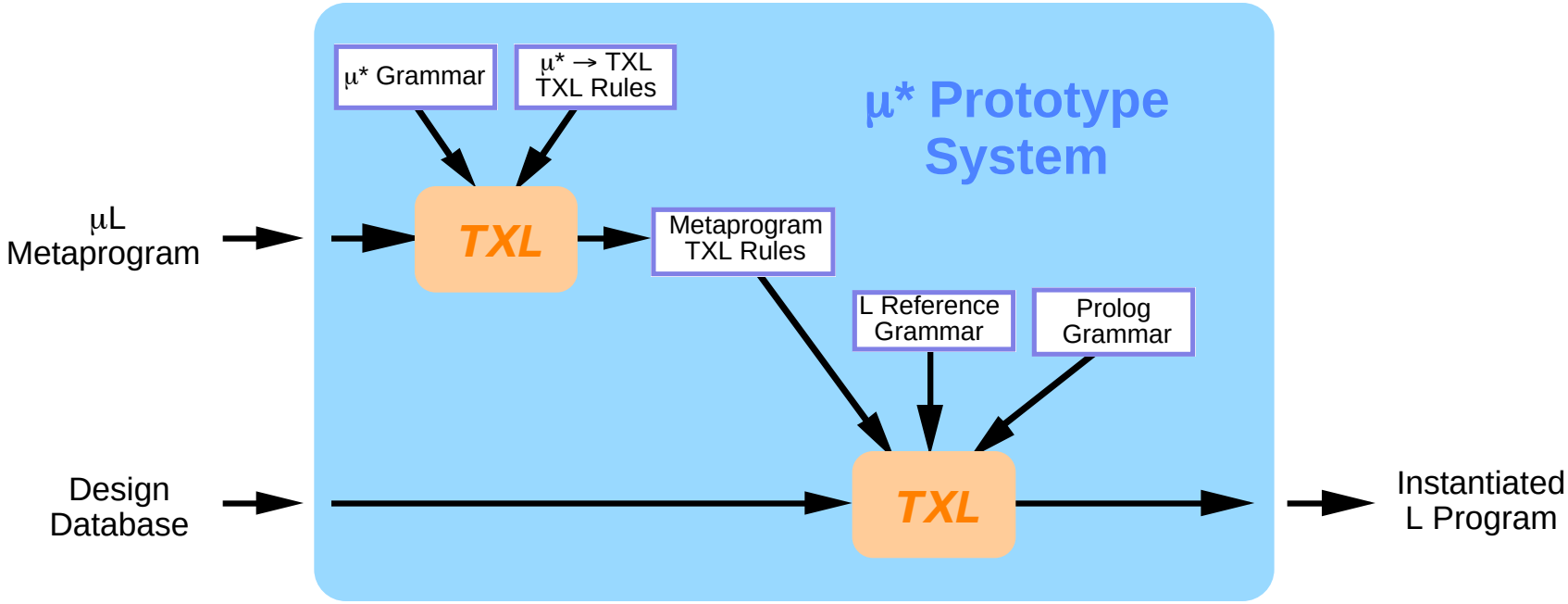
A Metaprogramming Language

- μ* is a family of metaprogramming languages with a common notation and implementation
- For example, μC is the metaprogramming dialect for C, μProlog the dialect for Prolog, and so on

```
\   struct {
        char *name;
        int (*addr)();
    } func[] =
    {
        $AllEntries,
        {"",0}
    };
\
    where AllEntries
            \ {$X,$Y} \ [list init]
        each function (F [id])
        where X \ "" \ [string] [" F]
        where Y \ mpro \ [id] [_ F]
\\
```

Implementing μ*

- Metaprogram first source transformed into a TXL ruleset
- The TXL ruleset is then itself run as a source transformation to instantiate the metaprogram



Metaprogramming Example

- Generation of interface code from SGI GL design spec

```
function (zbuffer).
returns (zbuffer, int).
function (color).
parameter (color, index, int, in).
function (winopen).
parameter (winopen,name,"char*",in).
returns (winopen, gid, int).
  . . .
```

GL Interface Design Specification

```
extern int mpro_zbuffer();
extern void mpro_color();
extern gid mpro_winopen();
   . . .
```

External C Routine Declarations for C Glue Routines

```
external (zbuffer/1,-n).
external (color/1,+n).
external (winopen/2,+s,-n).
  . . .
```

Prolog External Declarations

```
int mpro_zbuffer(p)
   struct {
      Boolean *b;
   } *p;
{
   Boolean b;
   b = (Boolean) *p -> b;
   zbuffer(b);
   return(0);
}
  . . .
```

C Glue Routines

```
struct {
   char name[];
   int (*addr)();
} func[] =
{
   {"arc", mpro_arc},
   {"arcf", mpro_arcf},
        . . .
   {"zclear", mpro_zclear},
   {"", 0}
};
```

C Entry Point Array

Metaprogramming Example

- Step 1: Transform metaprogram source to generate TXL ruleset

```
\    struct {
         char *name;
         int (*addr)();
     } func[] =
     {
         $AllEntries,
         {"",0}
     };
\
     where AllEntries
             \ {$X,$Y} \ [list init]
         each function (F [id])
         where X \ "" \ [string] [" F]
         where Y \ mpro \ [id] [_ F]
\\
```

µC Metaprogram

```
function execMetaProgram
    replace [repeat external_facts]
        N1 [repeat external_facts]

    construct AllEntries [list initializer
        _ [addAllEntries each N1]

    by
        struct {
            char * name ;
            int (* addr )();
        } func '[ '] =
        {
            AllEntries,
            {“”,0}
        };
end function
```

.
.
.

TXL Ruleset

Metaprogramming Example

- Step 2: Run generated TXL ruleset to transform design database to instantiation of metaprogram

```
function(addtopup).
parameter(addtopup,in,long,pup).
parameter(addtopup,in,String,str).
parameter(addtopup,in,long,arg).

function(arc).
parameter(arc,in,Coord,x).
parameter(arc,in,Coord,y).
parameter(arc,in,Coord,radius).
parameter(arc,in,Angle,startang).
parameter(arc,in,Angle,endang).

    . . . (750 other design facts)

function(zbuffer).
parameter(zbuffer,in,Boolean,b).

function(zclear).
```

GL Interface Design Specification

```
struct {
    char * name;
    int (* addr) ();
} func [] =
{
    {"addtopup", mpro_addtopup},
    {"arc", mpro_arc},
    {"arcf", mpro_arcf},
    {"arcfi", mpro_arcfi},
    {"arcfs", mpro_arcfs},
    {"arci", mpro_arci},
    {"arcs", mpro_arcs},

      . . .  (200 more entries)

    {"zbuffer", mpro_zbuffer},
    {"zclear", mpro_zclear},
    {"", 0}
};
```

C Entry Point Array

2nd Order Transformation

- In essence, what we have is metaprogramming as a 2nd order source transformation
- Q: Does this idea generalize to implementation of any generative system as a source transformation ?