NLPAPI - an API for Nonlinear Programming Problems

Michael E. Henderson
IBM Research

This started as part of a project to optimize circuit designs

EINSTIMER - given a circuit, finds delays and signal props.
LANCELOT - Nonlinear Optimizer (Trust Region).

so use LANCELOT to minimize delay (or power, or ...) and use EINSTIMER to evaluate the objective.

What's so hard about that?

1. EINSTIMER evaluates constraint values (limits on signal properties) and objective (e.g. delay through the circuit) algorithmically.
   No analytic statement of problem!

   LANCELOT uses a file based input (SIF).
   Requires an analytic statement of problem!

2. LANCELOT uses "Group Partially Separable" structure.
   Requires a particular form of input (complicated).

   EINSTIMER developers didn't know about "Group Partial Separability". (And even if they did...)
   Problem not stated in "correct" format.

3. LANCELOT has to be in charge (main program)

   EINSTIMER has to be in charge.

What was done.

LANCELOT has "external functions" that can be coded in FORTRAN
at the bottom of the SIF file -- and --

A LANCELOT developer was "in-house".

So...

Code a dummy FORTRAN routine in the SIF file which calls
EINSTIMER.
Have initialization code (EINSTUNER)
-read the circuit description.
-write a SIF file
-run the SIF decoder
-compile the output (four routines elfuns,...)
-link EINSTIMER and LANCELOT.
-invoke LANCELOT, which calls the dummy routine which
calls EINSTIMER.

Yuck.

What to do?

1. Keep your mouth shut. (Recommended).

2. Go around telling everyone that this is a crummy design.

What did we do?

Created a subroutine interface to LANCELOT.

Insulate the user (EINSTUNER) from the input file format.

Provide an API - application programming interface, so the problem can be supplied as a subroutine, without an analytic form.

What did we do?

NLPAPI "Nonlinear Programming API"

Two "phases" -

Define the problem.

Invoke the optimizer.

LANCELOT's input format:
Group Partially Separable Functions

$$G(\mathbf{x}) = \frac{1}{s_0} g_0\left(\omega_{00} f_{00}(R_{00}\mathbf{x}) + \ldots + \omega_{0n_0 0n_0} f_{0n_0}(R_{0n_0}\mathbf{x}) + \langle a_{0n_0}, \mathbf{x} \rangle - b_0 \right) + \ldots$$

$$+ \frac{1}{s_m} g_m\left(\omega_m f_{m0}(R_{m0}\mathbf{x}) + \ldots + \omega_{mn_m mn_m} f_{mn_m}(R_{mn_m}\mathbf{x}) + \langle a_{mn_m}, \mathbf{x} \rangle - b_m \right)$$

Problem variables

$\mathbf{x}$ in IR^N

G in IR

Group functions

g:IR → IR

f:IR^m → IR

Internal variables

Element functions

Range xfrms

R mxN matrices

m<<N

Linear elements

a N-vectors

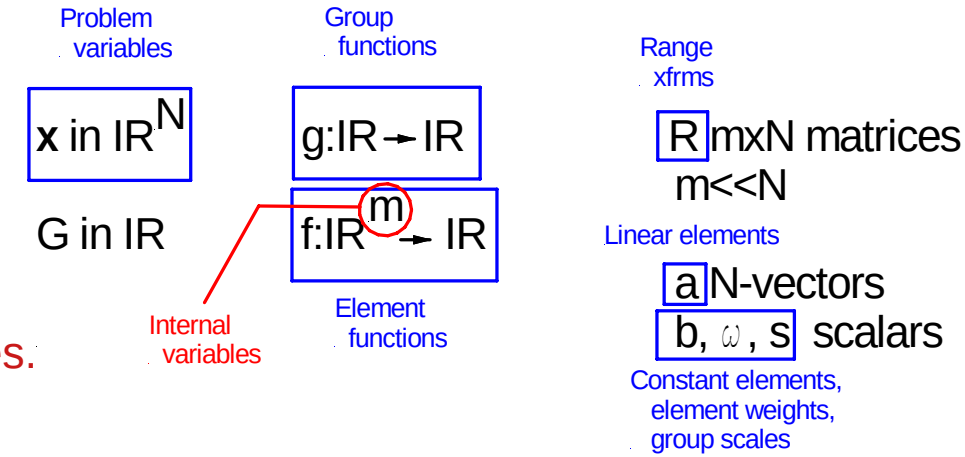
b, ω, s scalars

Constant elements, element weights, group scales

LANCELOT's input format:
Group Partially Separable Functions

$$G(\mathbf{x}) = \frac{1}{s_0} g_0\left(\omega_{00} f_{00}(R_{00}\mathbf{x}) + \ldots + \omega_{0n_0} f_{0n_0}(R_{0n_0}\mathbf{x}) + \langle a_{0n_0}, \mathbf{x} \rangle - b_0 \right) + \ldots$$

$$+ \frac{1}{s_m} g_m\left(\omega_m f_{m0}(R_{m0}\mathbf{x}) + \ldots + \omega_{mn_m} f_{mn_m}(R_{mn_m}\mathbf{x}) + \langle a_{mn_m}, \mathbf{x} \rangle - b_m \right)$$



Why?

Element functions f depend on only a "few" variables.

Derivatives of G can be expressed in terms of derivatives of g's ("Group Functions") and f's

Can reuse f and g's.

Makes linear dependancies (a's, b's -- "Linear Elements") explicit.

Drives users crazy

LANCELOT's "Nonlinear Programming Problem"

minimize **O**(**x**) Objective

$l_i < x_i < u_i$ Simple Bounds

$\mathbf{E}_i(\mathbf{x}) = 0$ Equality Constraints

$L_i < \mathbf{I}_i(\mathbf{x}) < U_i$ Inequality Constraints

O, E's, I's are all these "Group Partially Separable functions"

$$G(\mathbf{x}) = \frac{1}{s} g\left(\omega_0 f_0(R_0\,\mathbf{x}) + \ldots + \omega_n f_n(R_n\,\mathbf{x}) + \langle a_n, \mathbf{x} \rangle - b \right)$$

LANCELOT's "Nonlinear Programming Problem"

$$\text{minimize } \mathbf{O}(\mathbf{x}) + \mu \sum_i (\mathbf{E}_i(\mathbf{x}))^2$$

$$l_i < x_i < u_i$$ Simple Bounds

$$L_i < \mathbf{I}_i(\mathbf{x}) < U_i$$ Inequality Constraints

O, E's, I's are all these "Group Partially Separable functions"

$$G(\mathbf{x}) = \frac{1}{s} g\left(\omega_0 f_0(R_0\,\mathbf{x}) + \ldots + \omega_n f_n(R_n\,\mathbf{x}) + \langle a_n, \mathbf{x} \rangle - b \right)$$

except, LANCELOT requires that for E's and I's, g(x)=x

NLPAPI

A set of C routines for defining NLP's and solving them.

Basic design:

- Create "things".
- Use them, modify them.
- Free them.

NLPAPI

A set of C routines for defining NLP's and solving them.

Basic design:

- Create "things".
- Use them, modify them.
- Free them.

"things" are

Problems, which consists of lists of

- Objective(s?)
- Equality Constraints
- Element Functions
- Nonlinear Elements
- Simple Bounds
- Inequality Constraints
- Group Functions
- Groups

NLPAPI

A set of C routines for defining NLP's and solving them.

Basic design:

- Create "things".
- Use them, modify them.
- Free them.

"things" are

Problems, Solvers (LANCELOT, IPOPT)

- Set/Get parameters
- "minimize", "maximize"

NLPAPI

A set of C routines for defining NLP's and solving them.

Create "things".
Use them, modify them.
Free them.

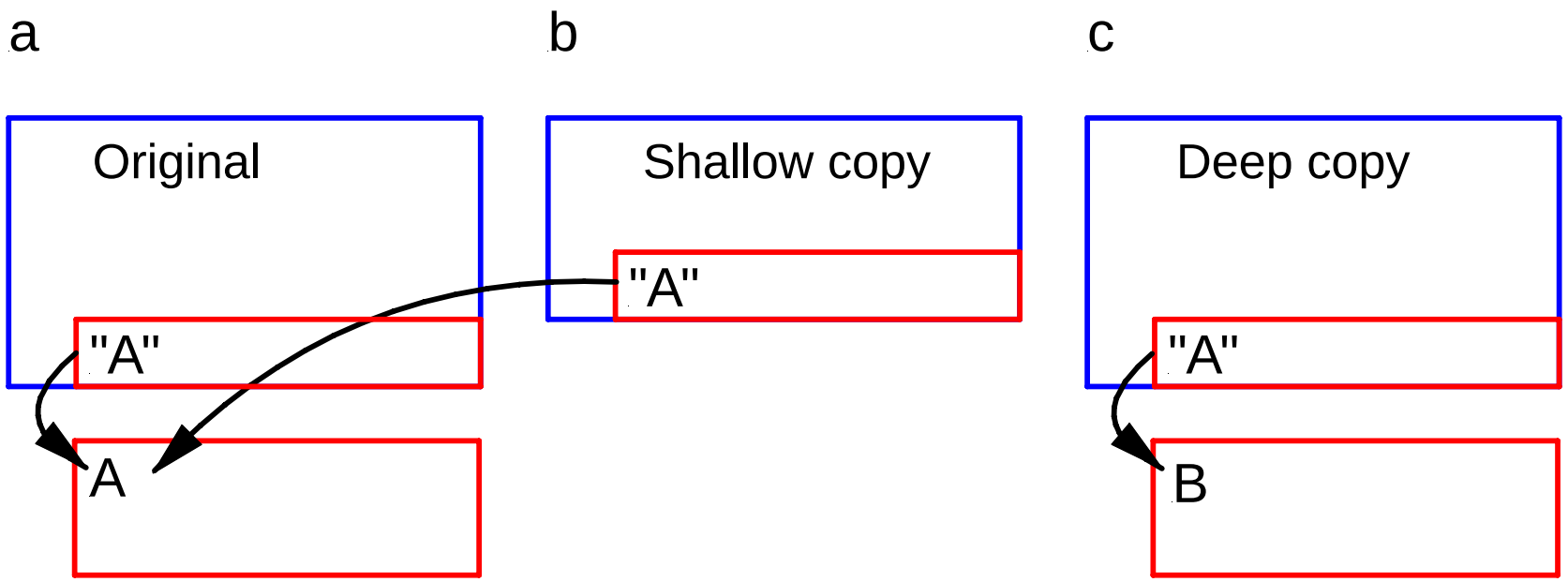
"things" are

Problems, Solvers

Managing "things"

Shallow copies/Deep copies

a,b,c are e.g. groups and
A is a nonlinear element
"A" is the name of a pointer

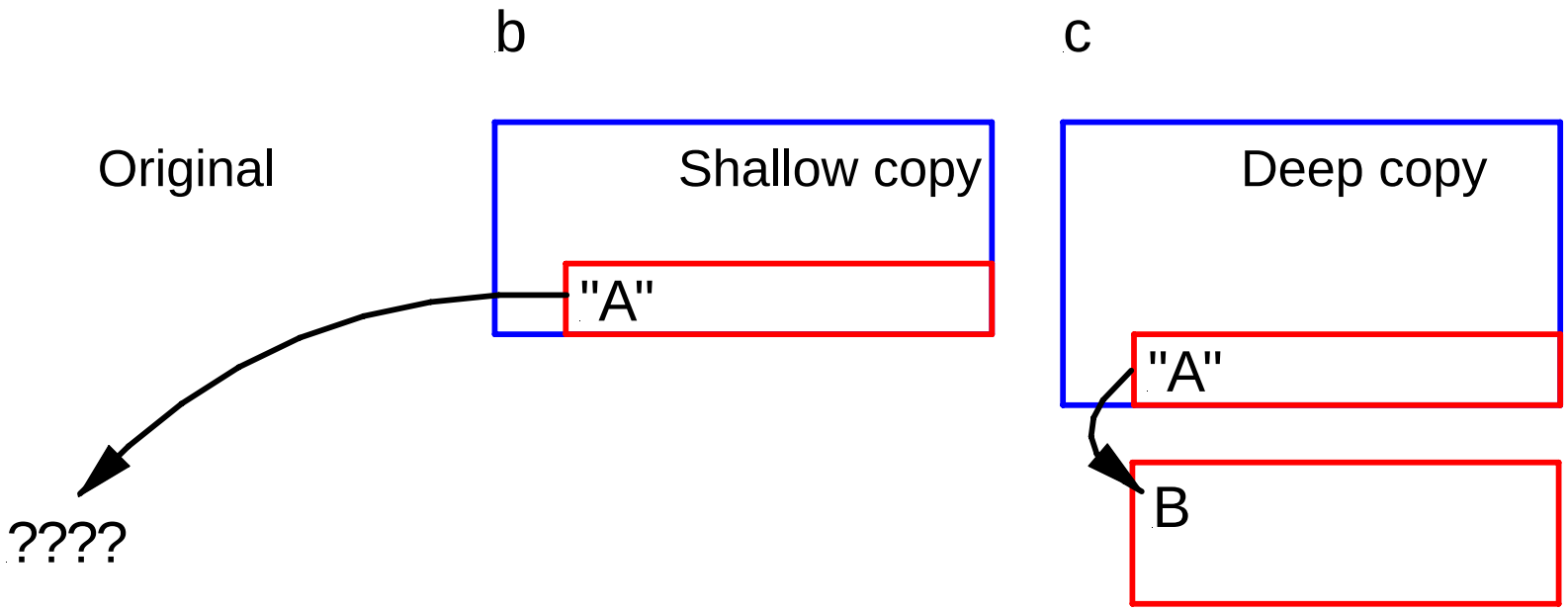


Reuse things: make shallow copies, not deep.

Managing "things"

Shallow copies/Deep copies

a,b,c are e.g. groups and
A is a nonlinear element
"A" is the name of a pointer

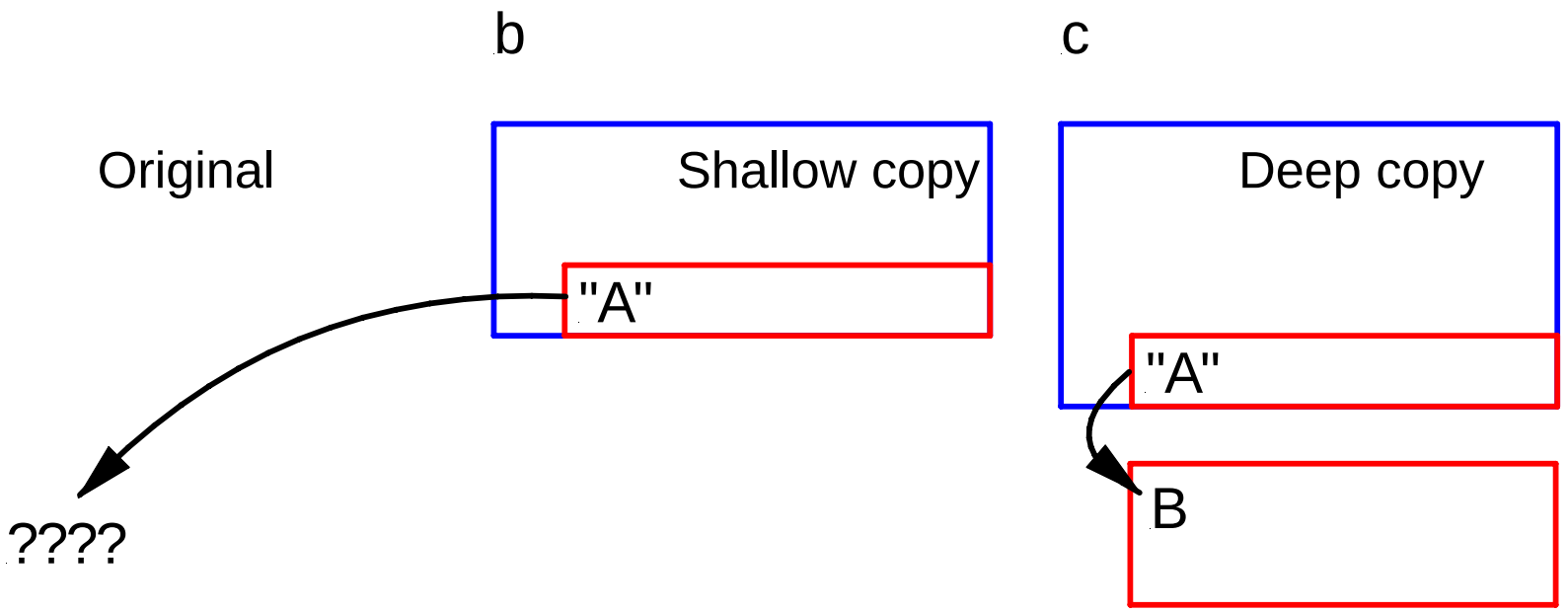


What happens to the shallow copy when the original disappears?

Managing "things"

Shallow copies/Deep copies

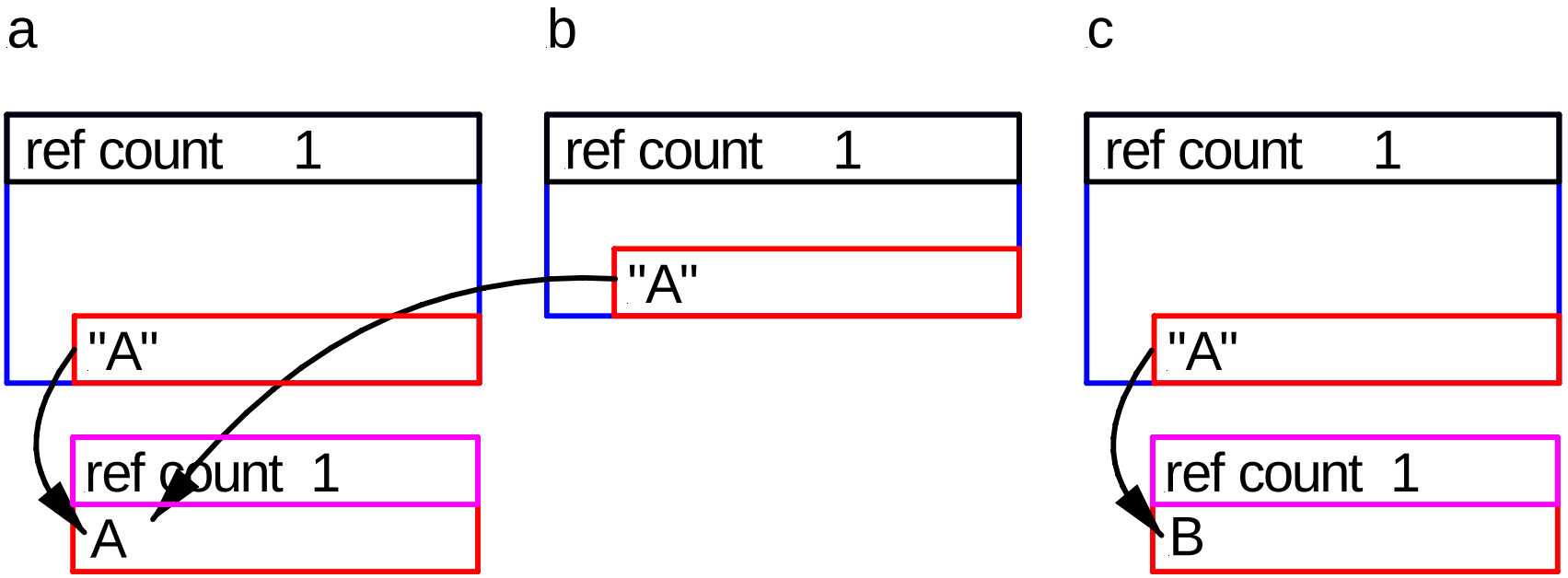
a,b,c are e.g. groups and
A is a nonlinear element
"A" is the name of a pointer



Solution: reference counting

Reference counting

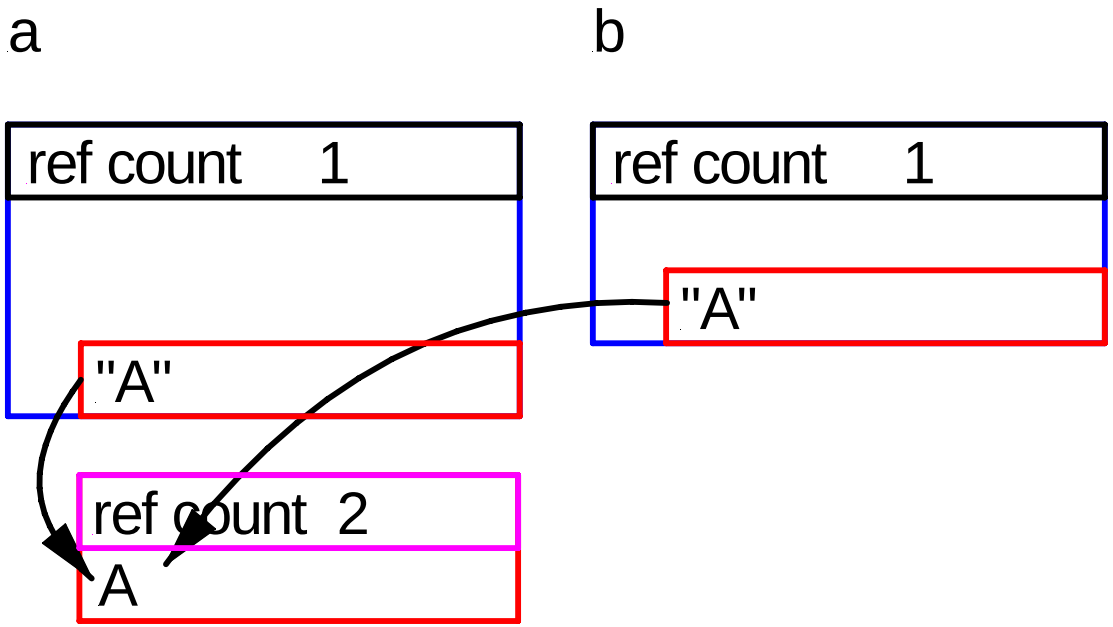
Add a "reference count" to all objects



When "a" creates "A" its ref count is 1 .

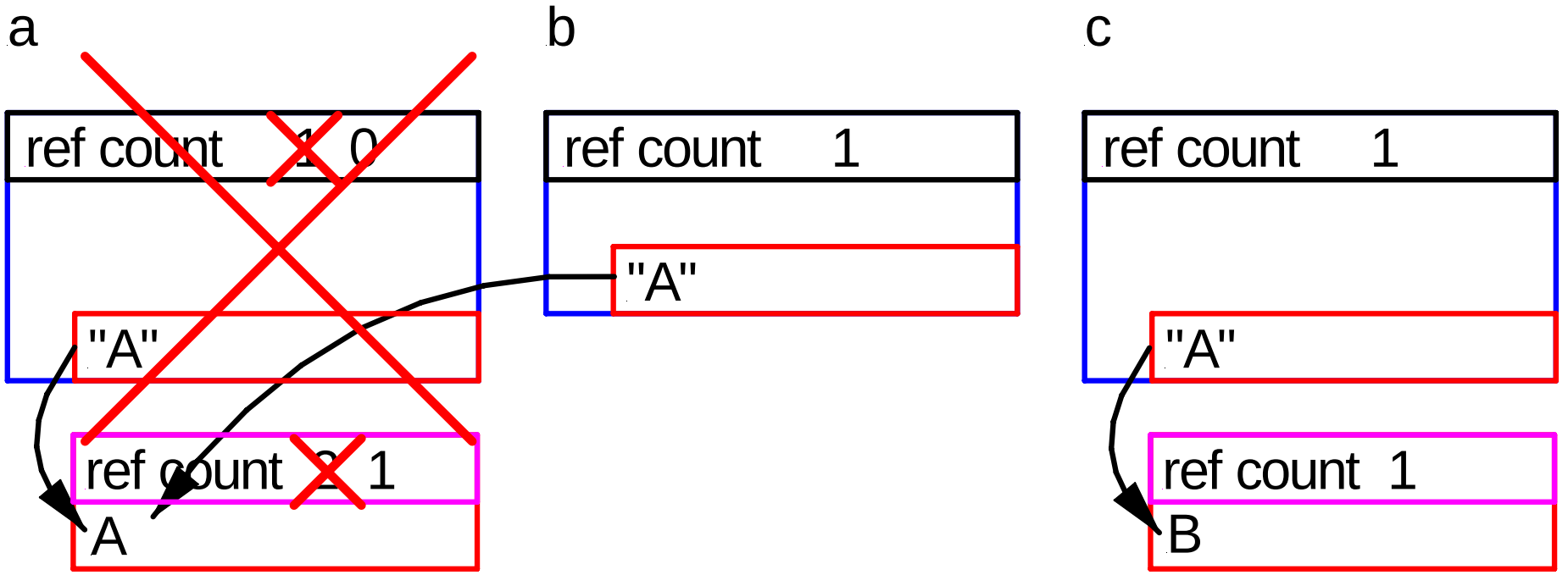
Instead of deleting "A", free it (decrement ref count), if the count is less than 1, delete it.

Reference counting



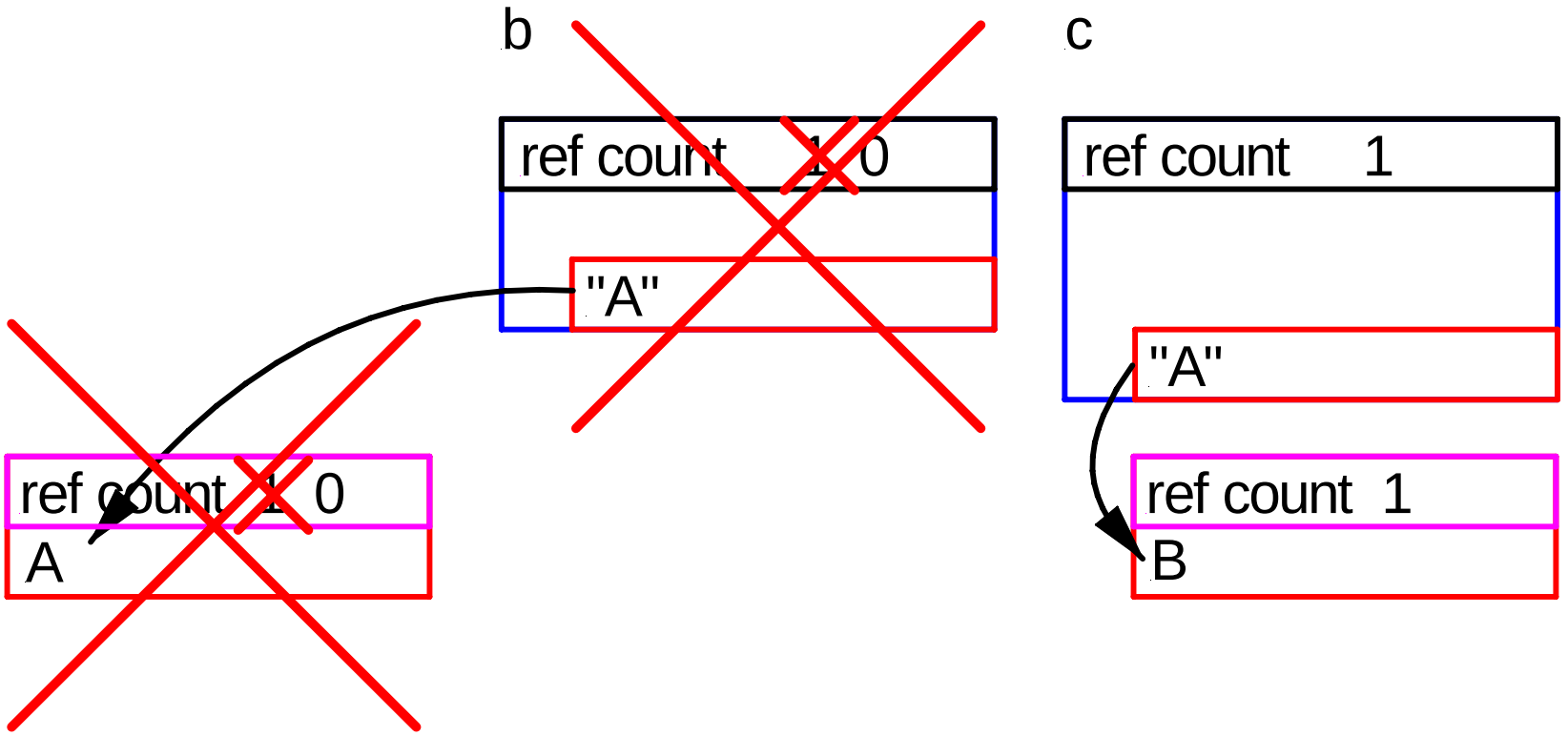
When a shallow copy is made increment the original's ref count.

Reference counting



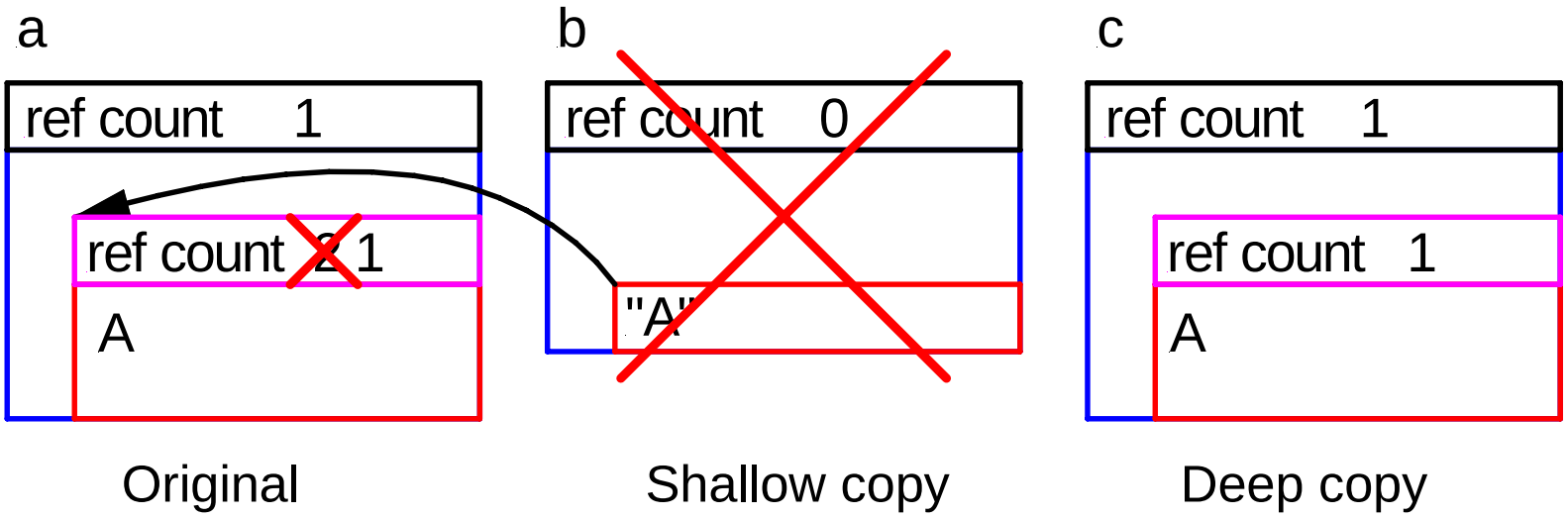
When a is deleted, "A" sticks around, but looses a count.

Reference counting



When the shallow copy is "free"ed, the original is deleted.

Reference counting




```
P=LNPCreateProblem("HS65",3);
```

bunch of stuff

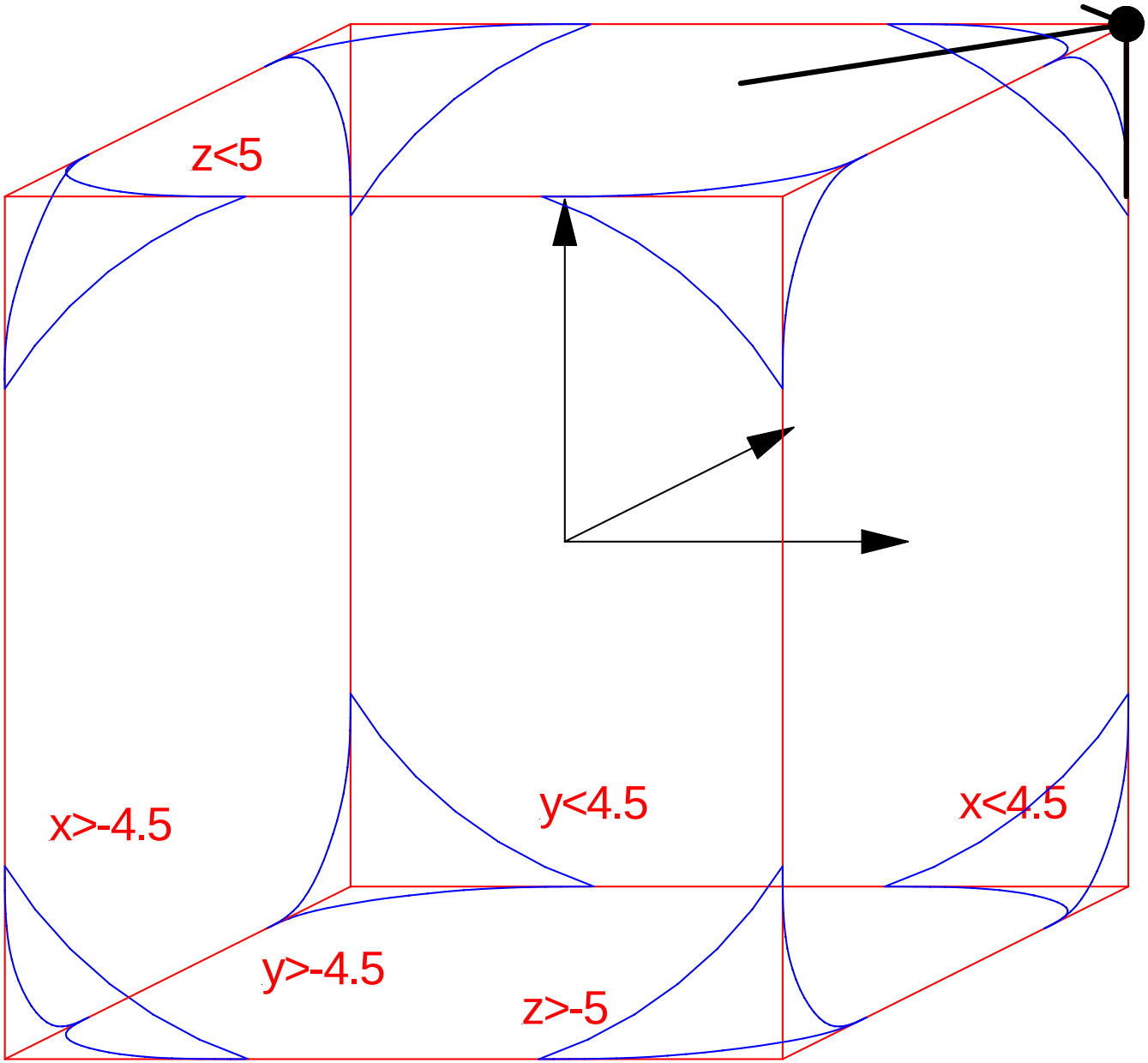
```
NLPFreeProblem(P);
```

P doesn't necessarily disappear here

Example

HS 65

Min (x-y)**2+(x+y-10)**2/9+(z-5)**2



-4.5<x<4.5
-4.5<y<4.5
-5 <z< 5

x**2+y**2+z**2<=48

Code -- not using groups

```
#include <NLPAPI.h>
#include <stdio.h>

int main(int argc,char *argv[])
 {
  LNProblem P;
  int nv=3;
  int v[3]={0,1,2};

  P=LNPCreateProblem("HS65",3);
  NLPSetSimpleBounds(P,0,-4.5,4.5);
  NLPSetSimpleBounds(P,1,-4.5,4.5);
  NLPSetSimpleBounds(P,2,-5,5);

  NLPSetObjectiveByString(P,"obj",
                                  nv,v, "[x,y,z]",
                                  "(x-y)**2+(x+y-10)**2/9+(z-5)**2");

  NLPAddInequalityConstraintByString(P,"ineq0",
                                           -1.e30,0.,
                                           nv,v, "[x,y,z]",
                                          "x**2+y**2+z**2-48");

  NLPrintProblemShort(stdout,P);
  NLPFreeProblem(P);
 }
```

HS65

Simple Bounds:

-4.5000000 <=X1 <= 4.5000000
-4.5000000 <=X2 <= 4.5000000
-5.0000000 <=X3 <= 5.0000000

Objective Function:

(x-y)**2+(x+y-10)**2+(z-5)**2

Inequality Constraints:

0 ineq0
x**2+y**2+z**2-48<0.000000

Groups

Objective

Min $(x-y)^{**}2 + (x+y-10)^{**}2/9 + (z-5)^{**}2$ 3 Groups, no Nonlinear Elements

$$\frac{1}{s} g(\langle a,x\rangle) + \frac{1}{s} g(\langle a,x\rangle - b) + \frac{1}{s} g(\langle a,x\rangle - b)$$

Simple Bounds on Variables

$-4.5 < x < 4.5$
$-4.5 < y < 4.5$
$-5 < z < 5$

Inequality Constraint (for LANCELOT group function must be identity and single group)

$x^{**}2 + y^{**}2 + z^{**}2 <= 48$ 1 Group, 3 Nonlinear Elements

$$\frac{1}{s} g(\omega f(R\mathbf{x}) + \omega f(R\mathbf{x}) + \omega f(Rx) - b) < 0$$

Code -- for those who like groups

```
NLGroupFunction g;
NLElementFunction f;

g=NLCreateGroupFunctionByString(P,"gsq","[x]","x**2");

NLPAddGroupToObjective(P,"obj0");
NLPAddGroupToObjective(P,"obj1");
NLPAddGroupToObjective(P,"obj2");

NLPSetObjectiveGroupFunction(P,0,g);
NLPSetObjectiveGroupFunction(P,1,g);
NLPSetObjectiveGroupFunction(P,2,g);

a=NLCreateVector(3);NLVSetC(a,0,1);NLVSetC(a,1,-1);
NLPSetObjectiveGroupA(P,0,a);
NLFreeVector(a);

a=NLCreateVector(3);NLVSetC(a,0,1);NLVSetC(a,1,1);
NLPSetObjectiveGroupA(P,1,a);
NLFreeVector(a);
NLPSetObjectiveGroupB(P,1,10);

        ...

constraint=NLPAddNonlinearInequalityConstraint(P,"ineq0");
NLPSetInequalityConstraintUpperBound(P,constraint,0.);
NLPUnSetInequalityConstraintLowerBound(P,constraint);
NLPSetInequalityConstraintGroupB(P,constraint,0,48.);

f=NLCreateElementFunctionByString(P,"esq",1,(NLMatrix)NULL,"[x]","x**2");
v[0]=0;
ne=NLCreateNonlinearElement(P,"Sq1",f,v);
element=NLPAddNonlinearElementToInequalityConstraintGroup(P,constraint,0,1.,ne);
NLFreeNonlinearElement(P,ne);
```

HS65

Simple Bounds:

-4.5000000 <=X1 <= 4.5000000
-4.5000000 <=X2 <= 4.5000000
-5.0000000 <=X3 <= 5.0000000

Objective Function:

"gsq"(X1-X2)+"gsq"(X1+X2-10)+"gsq"(X3-5)

Inequality Constraints:

0 ineq0
x**2+x**2+x**2-48<0.000000

Objective

Min (x-y)**2 + (x+y-10)**2/9 + (z-5)**2

$\frac{1}{s}g(\langle a,x\rangle) + \frac{1}{s}g(\langle a,x\rangle - b) + \frac{1}{s}g(\langle a,x\rangle - b)$

Simple Bounds on Variables

-4.5<x<4.5
-4.5<y<4.5
-5 <z< 5

Groups
Nonlinear Elements

Inequality Constraint

x**2 + y**2 + z**2 <=48

$\frac{1}{s}g(\omega f(Rx) + \omega f(Rx) + \omega f(Rx) - b) < 0$

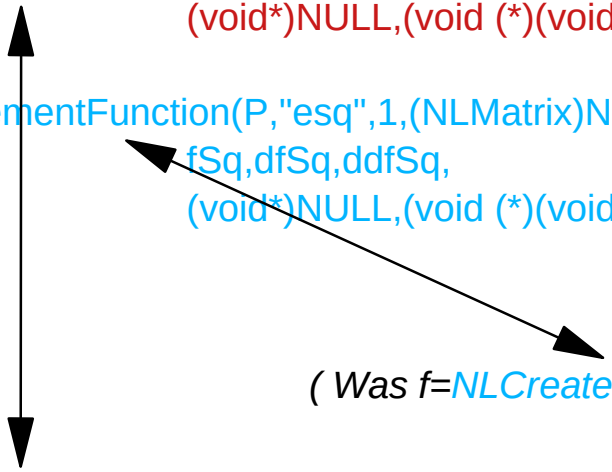
Where are EINSTIMER's subroutines?

```
double gSq(double x,void *d){return(x*x);}
double dgSq(double x,void *d){return(2*x);}
double ddgSq(double x,void *d){return(2);}

double fSq(int n,double *x,void *d){return(x[0]*x[0]);}
double dfSq(int i,int n,double *x,void *d){return(2*x[0]);}
double ddfSq(int i,int j,int n,double *x,void *d){return(2);}

g=NLCreateGroupFunction(P,"gsq",gSq,dgSq,ddgSq,
                        (void*)NULL,(void (*)(void*))NULL);

f=NLCreateElementFunction(P,"esq",1,(NLMatrix)NULL,
                          fSq,dfSq,ddfSq,
                          (void*)NULL,(void (*)(void*))NULL);
```



*(Was g=NLCreateGroupFunctionByString(P,"gsq","[x]","x**2");)*

*(Was f=NLCreateElementFunctionByString(P,"esq",1,(NLMatrix)NULL,"[x]","x**2");)*

HS65

Simple Bounds:

-4.5000000 <=X1 <= 4.5000000
-4.5000000 <=X2 <= 4.5000000
-5.0000000 <=X3 <= 5.0000000

Objective Function:

"gsq"(X1-X2)+0.111111*"gsq"(X1+X2-10)+"gsq"(X3-5)

Inequality Constraints:

0 C1
"esq"(X1)+"esq"(X2)+"esq"(X3)-48<0.000000

Solvers

```
NLLancelot Lan;

x0[0]=-5.;
x0[1]=5.;
x0[2]=0.;

Lan=NLCreateLancelot();
rc=LNSetPrintLevel(Lan,1);
rc=LNSetInitialPenalty(Lan,1.e-4);
rc=LNSetPenaltyBound(Lan,1.e-4);

rc=LNMinimize(Lan,P,x0,          initial guess
              (double*)NULL,     initial slacks
              (double*)NULL,     initial λ's
               x);               solution

NLFreeLancelot(Lan);
```

(3.650460,3.650460,4.620420)

```
NLIpopt Ip;

x0[0]=-5.;
x0[1]=5.;
x0[2]=0.;

Ip=NLCreateIpopt();
IPAddOption(Ip,"ioutput",1.);
IPAddOption(Ip,"dtol",1e-12);

rc=IPMinimize(Ip,P,x0,
              (double*)NULL,
              (double*)NULL,
               x);

NLFreeIpopt(Ip);
```

(3.650462,3.650462,4.620418)

Basics of NLPAPI

- a subroutine interface for stating and solving nonlinear optimization problems.
- a " for invoking NLP solvers
- a " for NLP solvers to access the NLP.

NLPAPI was built as an interface to LANCELOT, for tuning circuits.
Now also works with IPOPT.

Based on, not a general algebraic expression, but a particular "simple" representation ~ "automatic differentiation"

-Insulates the user from data structure/file format. (like a modeling lang.)
-Subroutines can be used to define the problem.
-When the optimizer is changed the "problem" doesn't have to change.

In the future

1. Something ~~better~~ more general than Group Partial Sep.

   Create "scalar functions" instead of group functions,
   "vector valued functions" instead of nonlinear elements

   Add, Mult., Compose, etc these functions.

   Can still find the sparsity structure and evaluate derivatives.

2. Interfaces via SWIG (http://sourceforge.net/projects/swig)

   generates wrappers for interfaces, and can be extended.

   comes with Python, Perl and other interpretive langs.
   I'm working on extensions for Matlab, Excel, DX.