Osprey:
A Practical Type System for Validating Dimensional Unit Correctness of C Programs

Lingxiao Jiang and Zhendong Su
University of California, Davis

Supported by National Science Foundation
WHERE DISCOVERIES BEGIN

Motivation

- Measurement units important in scientific applications

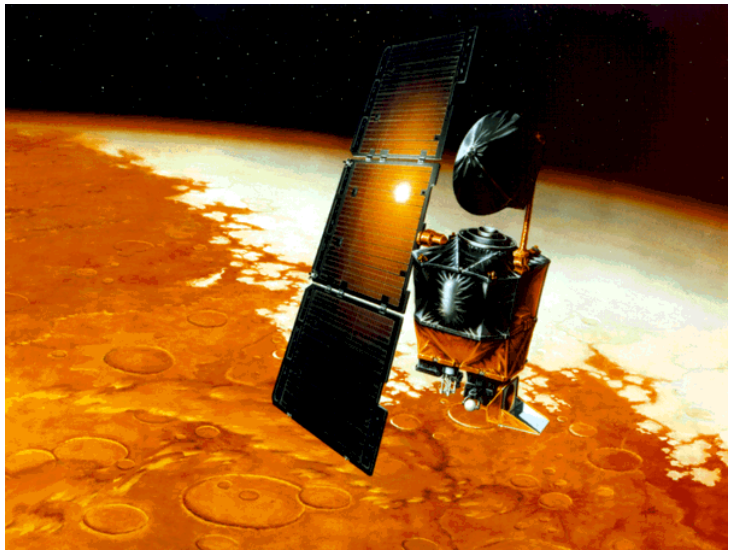
- There are many measurement units
  - 7 base units in International System of Units (SI units):
    - meter, kilogram, second, ampere, kelvin, mole, candela
  - 20 SI unit prefixes:
    - tera (10^{12}), giga(10^{9}), mega(10^{6}), deci(10^{-1}), centi(10^{-2}), …
  - English units:
    - inch, yard, acre, pound, gallon, …

Motivation

- Standard type systems don't enforce unit correctness
- Easy to make mistakes in large scale programs

Loss of Mars Climate Orbiter, 1999

- Data in English units and SI units mixed.
- Wrong navigation signals issued by software



Courtesy JPL & NASA

Motivation – Sample Error

- Easy to make mistakes with even small programs

```
double mile2meter(double x) {
    return ( x * 1682 );
}
```

Motivation – Sample Error

- Easy to make mistakes with even small programs

```
double mile2meter(double x) {
    return ( x * 1682 );
}                    1609
```

A Working Example

- $$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL_{rad}^{'}\}$$

- **Compute an electron's final energy after traversing a thickness of a given material**

```
extern double alpha, N_A; extern double r_e;

typedef struct {
    double atomicWeight;
    double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, double density,
                   double thick, double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

Existing Approaches

- Dimensional analysis
  - Both sides of an equation should have the same unit
  - Effective but manual
- Language type extensions
  - User-defined types to represent units and unit arithmetic
  - E.g., SIUNITS for C++, MetaGen for Java
  - Inconvenient for legacy code; limited expressiveness.
- Runtime checking
  - Unit Assertions
  - E.g., C-UNITS
  - Intolerable overhead; incomplete coverage

Our Approach

- Lightweight unit annotations

```
$meter double mt; $mile double ml;
double mile2meter($mile double x)
    return x * ($f)1682;
}
```

Our Approach

- Lightweight unit annotations

```
$meter double mt; $mile double ml;
double mile2meter($mile double x)
    return x * ($f)1682;
}
```

- Type inference detects possible inconsistencies

```
mt + ml;
mile2meter(mt);
mt = mile2meter(ml);
```

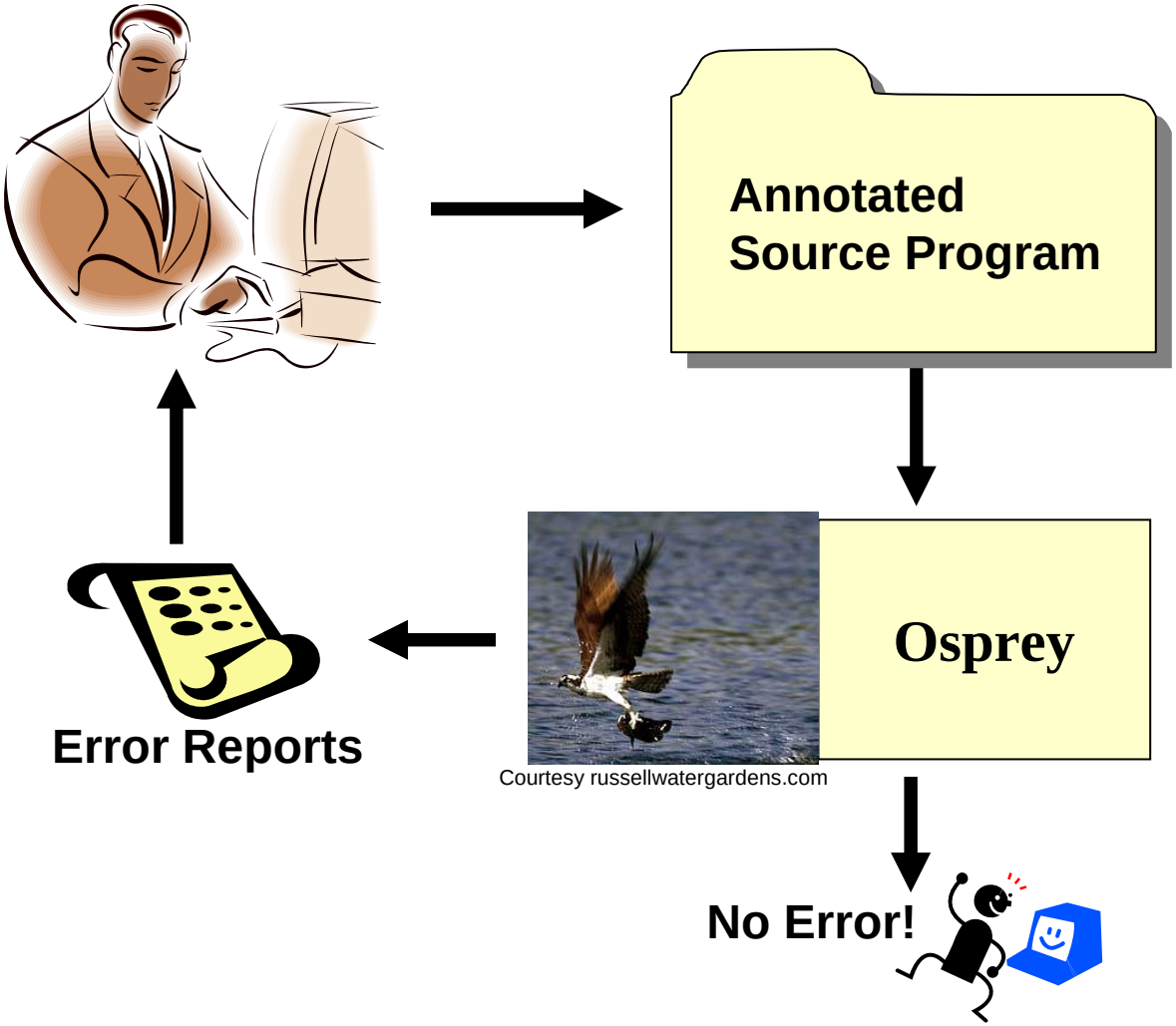
$$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL'_{rad}\}$$

```
extern        double alpha, N_A; extern    double r_e;
typedef struct {
        double atomicWeight;
           double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material,          double density,
                        double thick,           double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

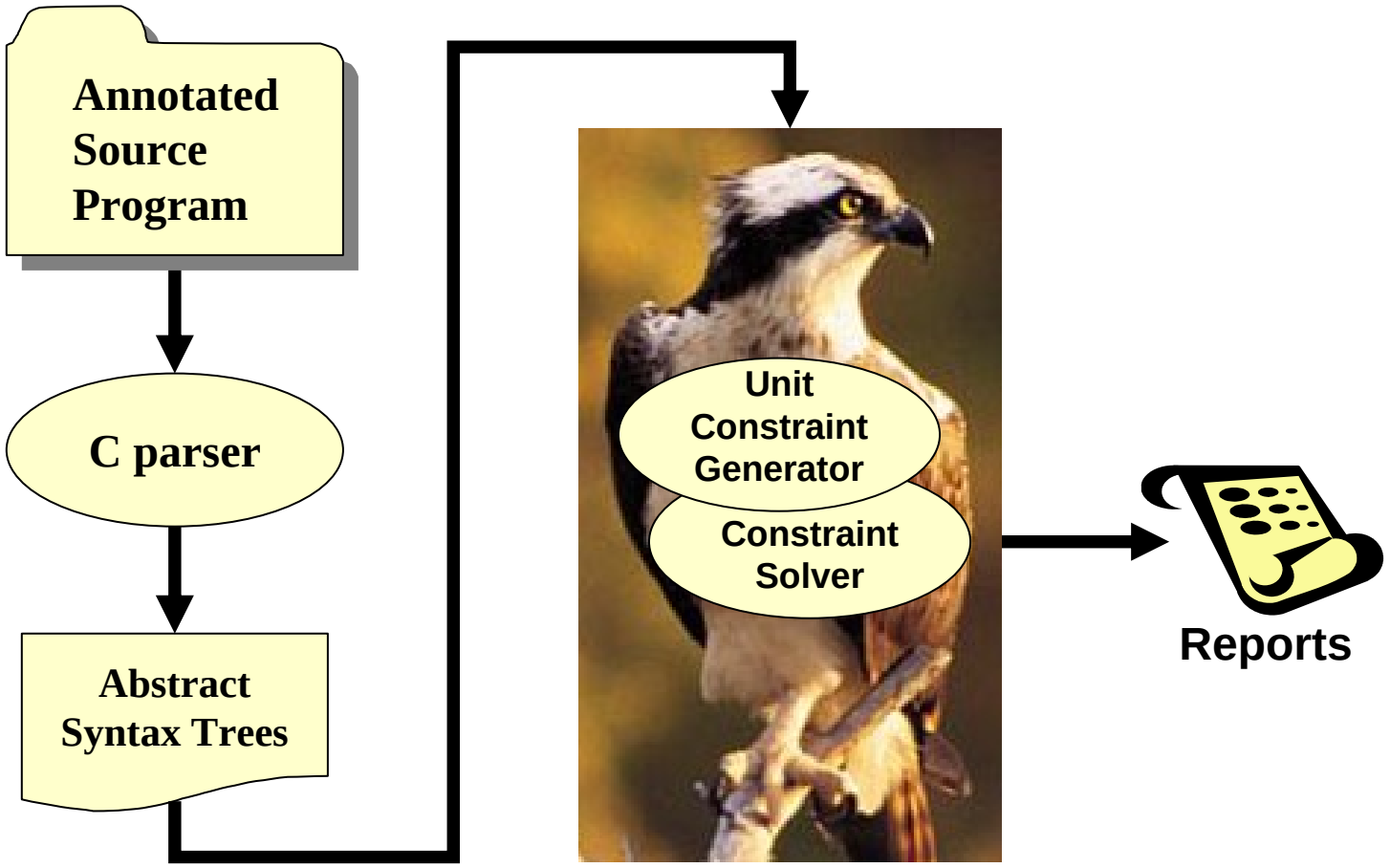
$$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL'_{rad}\}$$

```
extern $unity double alpha, N_A; extern $m double r_e;
typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

Users' View of Osprey

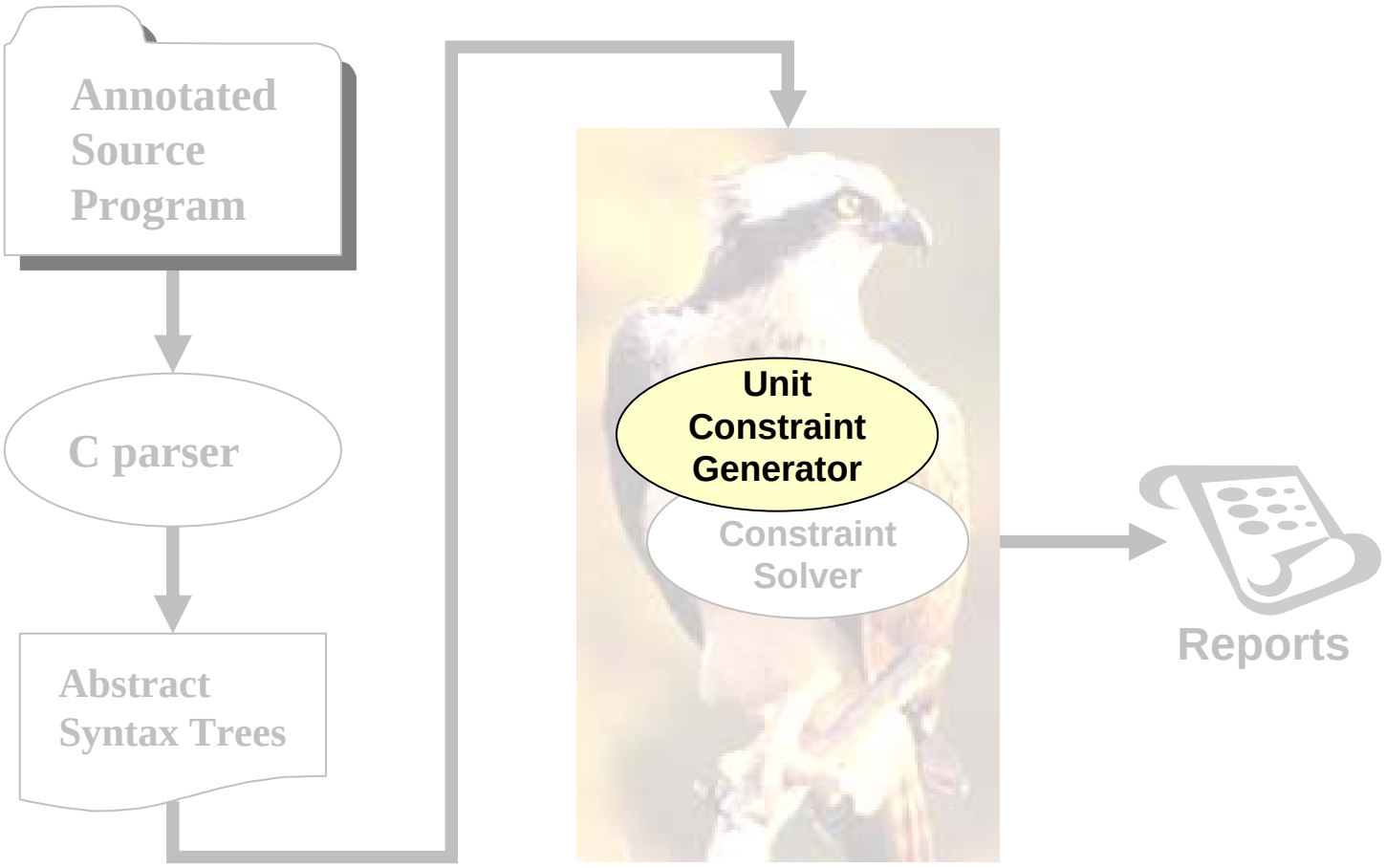


Osprey's Internal Structure



Courtesy ospreywatch.co.uk

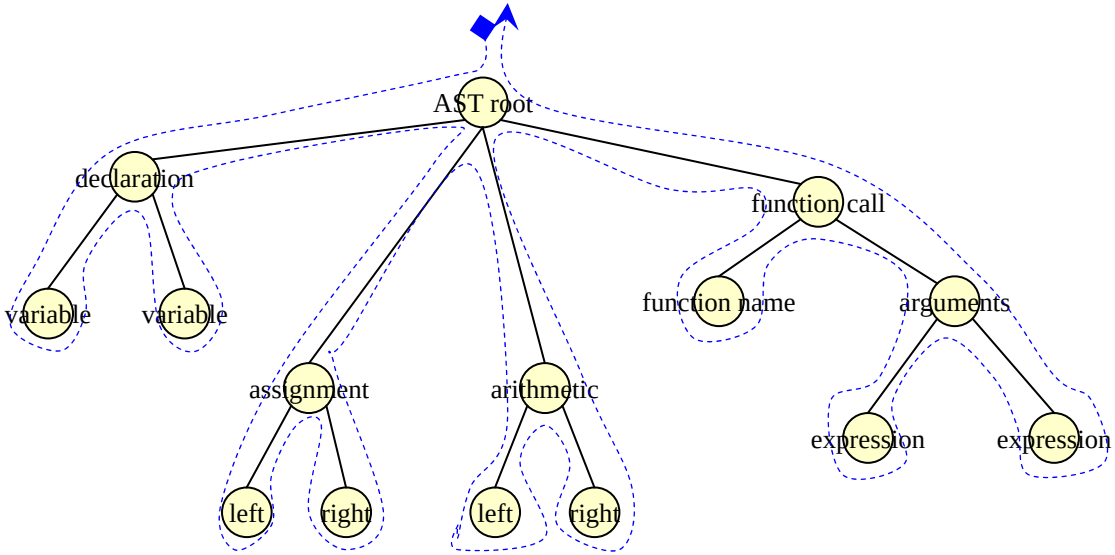
Osprey's Internal Structure



Courtesy ospreywatch.co.uk

Constraint Generation

- **Recursively traverse a program's AST and generate constraints w.r.t. types of AST nodes:**
  - **Declarations**
  - **Assignments**
  - **Arithmetic ops**
  - **Function calls**
  - **……**



```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Ele

doubl
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

`$unity double alpha;` → $u_{alpha} = unity$

```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    re
}

double                                                    nsity,
                    $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

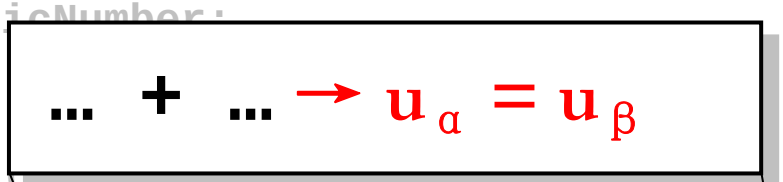
A = material->atomicWeight

→ $u_A = u_{atomicWeight}$

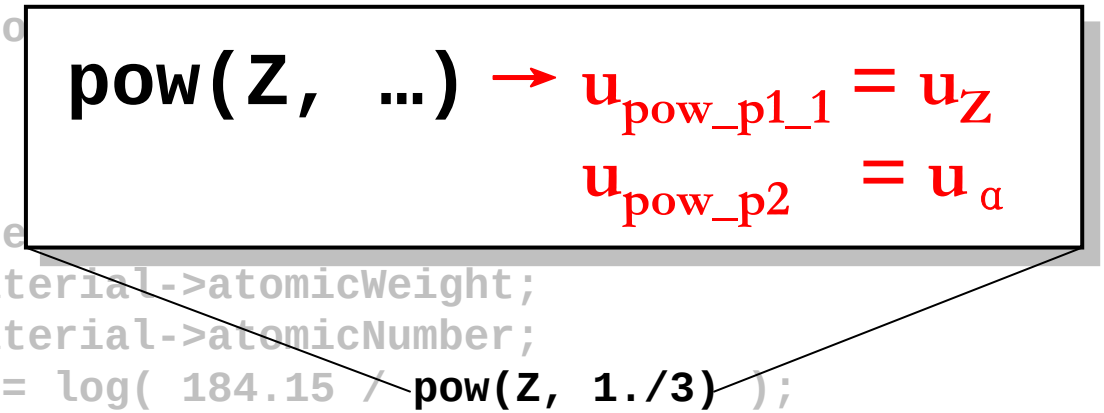
```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double atomicWeight;
    $unity dou
} Element;

double radiation
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

$$Z * L_{p\_rad} \rightarrow u_{\alpha} = u_Z * u_{Lp\_rad}$$

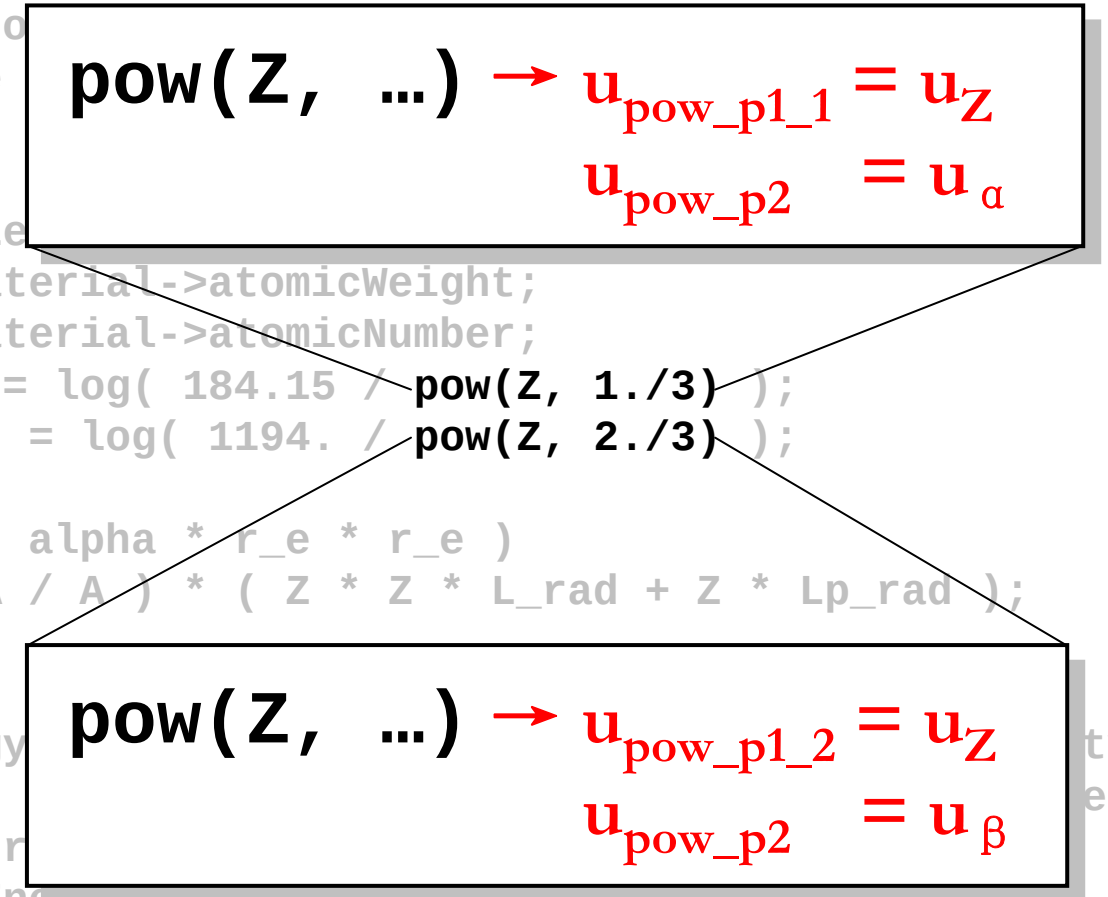
```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```



```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double ato
    $unity double
} Element;

double radiationLe
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```



```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double ato
    $unity double
} Element;

double radiationLe
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy                                    ty,
                                                      ergy){
    double X_0 = r
    return  initEnergy * exp( thick / X_0 );
}
```



```
extern $unity double alpha, N_A; extern $m double r_e;

typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );
    re
                                                    );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

X_0 = radiationLength…

→ $u_{X\_0} = u_{radiationLength\_return\_1}$

```
extern $unity double alpha, N_A; extern $m double r_e;

typedef str
    $kg dou
    $unity
} Element;

double radi
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = radiationLength(material);
    return  initEnergy / exp( thick / X_0 );
}
```

log(…) → $u_{log\_p1} = unity$

$u_{log\_return} = unity$

$u_{log\_p1} = u_{\alpha}$

```
extern $unity double alpha, N_A; extern $m double r_e;

typedef str
    $kg dou
    $unity
} Element;

double radi
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double fina                                          ble density,
                                                     le initEnergy){
    double
    return
}
```

$\log(\ldots) \rightarrow u_{log\_p1} = unity$

$u_{log\_return} = unity$

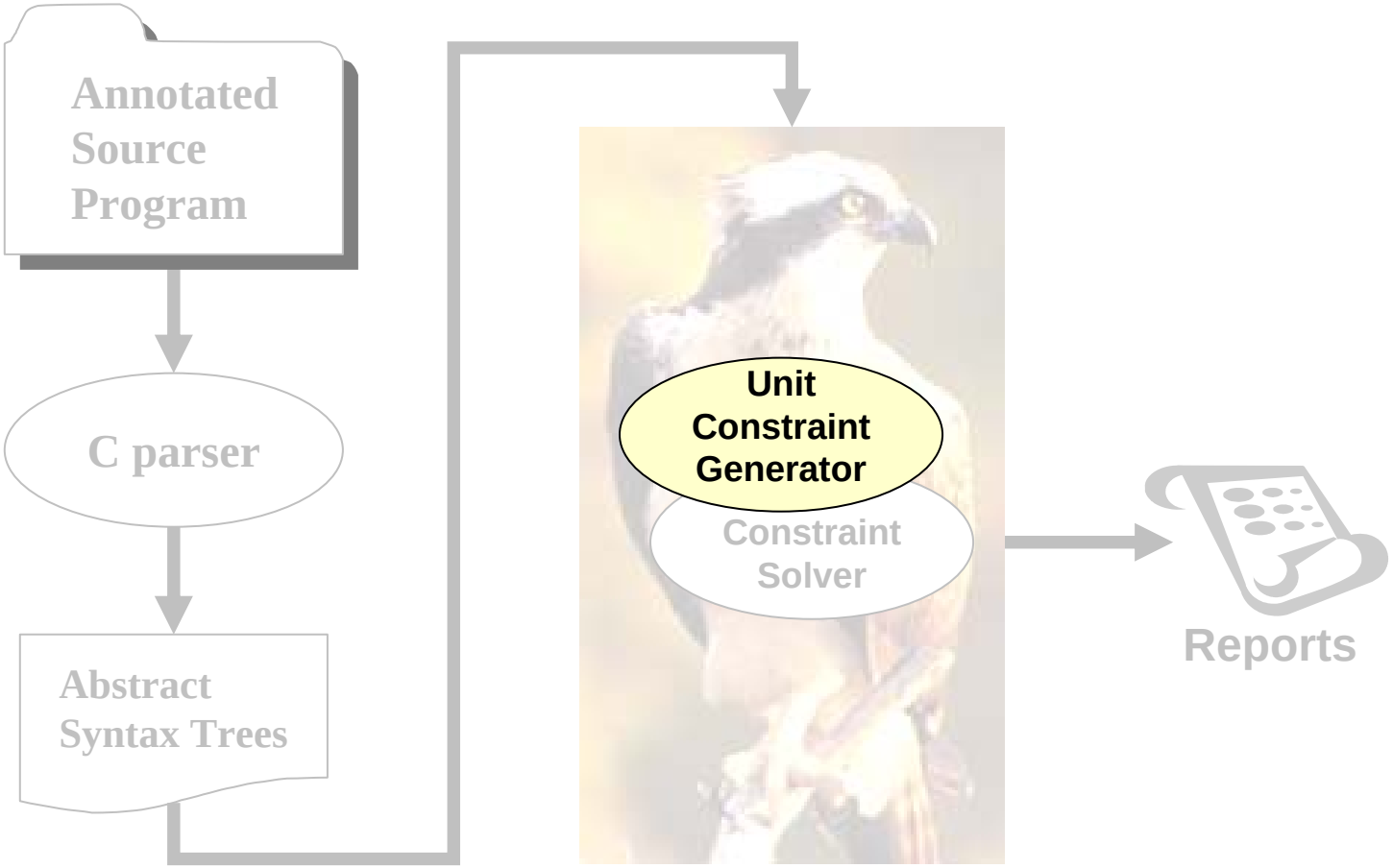
$u_{log\_p1} = u_{\alpha}$

$\log(\ldots) \rightarrow u_{log\_p1} = unity$

$u_{log\_return} = unity$

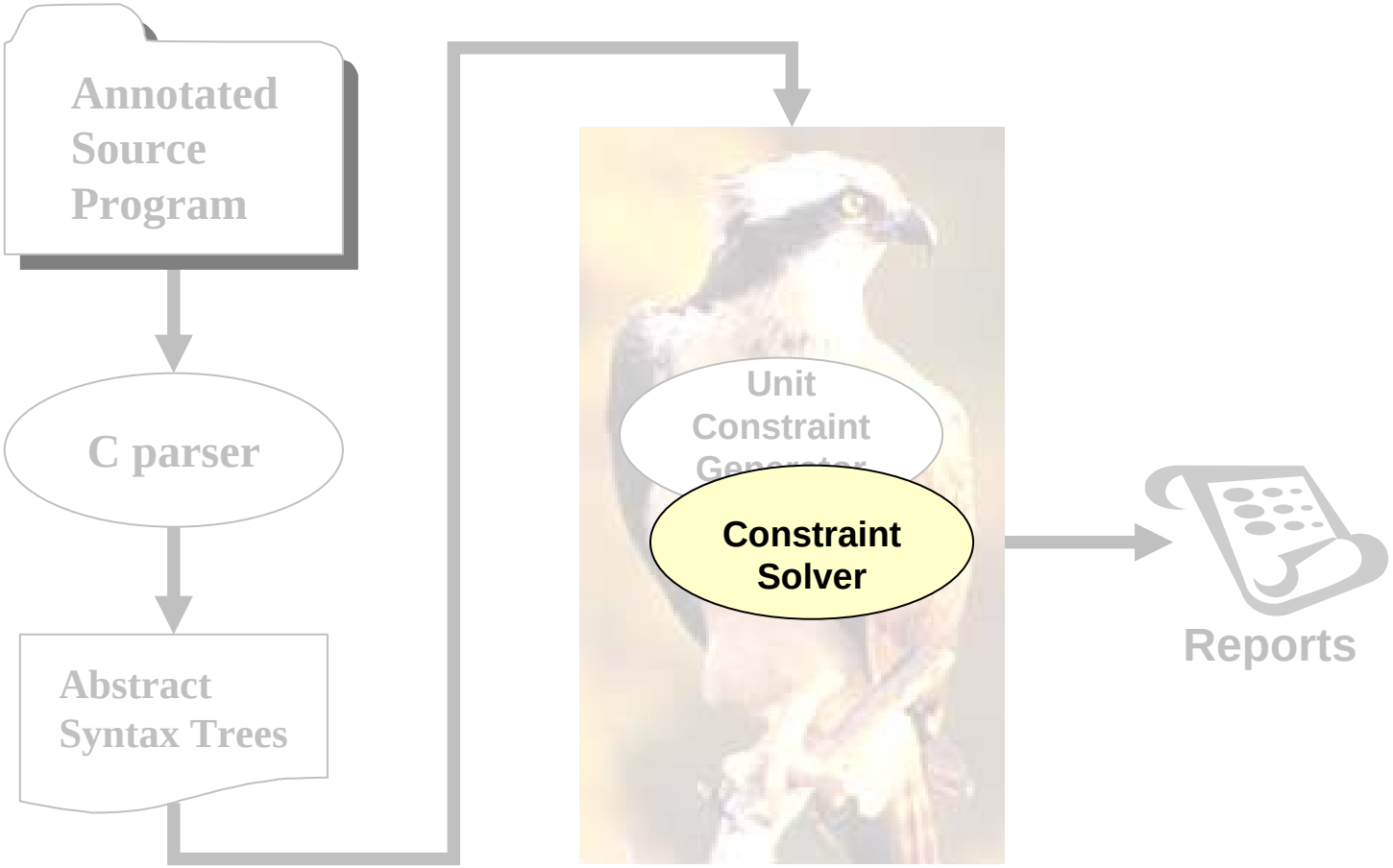
$u_{log\_p1} = u_{\beta}$

Osprey's Internal Structure



Courtesy ospreywatch.co.uk

Osprey's Internal Structure



Courtesy ospreywatch.co.uk

Constraint Solver - Equality Constraints

- Equality Constraints: $u_x = u_y$
- Banshee: solver engine for constraint-based program analyses
  - Find equivalent class representatives (ECRs), i.e., (partially) solution, for every variable.
  - NOT for multiplicative constraints: $u_x = u_y * u_z$

Constraint Solver-Multiplicative Constraints (1)

- Represent each unit with base units
  - `Energy=meter²kilogram¹second⁻² → [2,1,-2,0,0,0,0]`
  - `1 mile=1609*meter → [1,0,0,0,0,0,0]*1609`

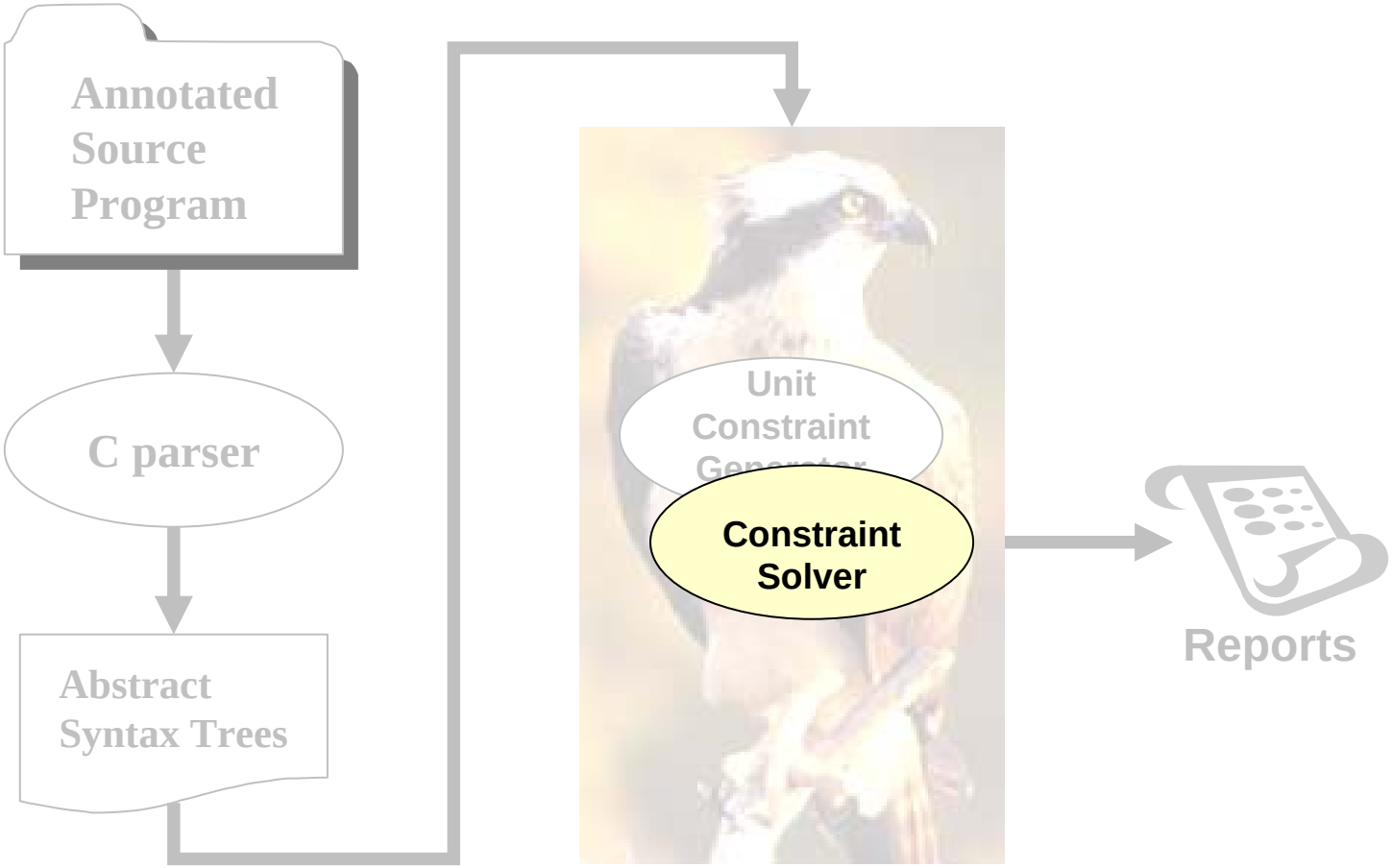
- Normalize constraints to the following form:
  - `u = v * w * f` // where f is a factor
  - Apply logarithm to transform them into linear equations:
    - $u_m = v_m + w_m$; // w.r.t. meter
    - …… // w.r.t. kilogram, second, ampere, kelvin, mole, candela
    - $u_f = v_f + w_f + \log_{10} f$; // w.r.t. unit conversion factor

Constraint Solver-Multiplicative Constraints (2)

- **Gaussian Elimination with CLAPACK**

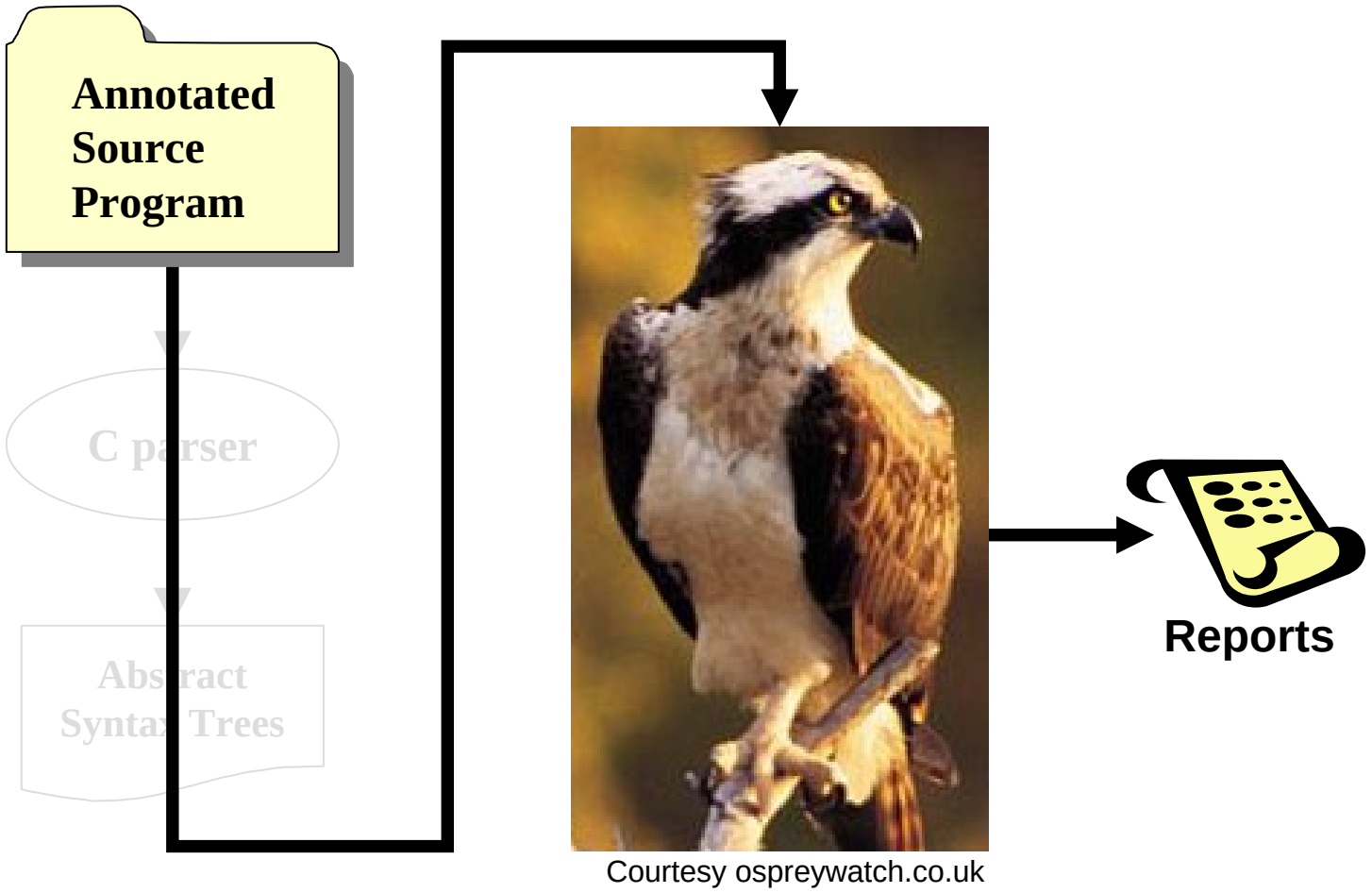
- **A specialized Union/Find algorithm for scalability:**
  - **E.g.,**
    - **suppose $ECR(u_1)=m$, $ECR(u_2)=m^2kg^{-1}$, then $u_1=u_2*u_3$ can be reduced to $kg*m^{-1}=u_3$**
    - **$u_4*u_4=u_4$ can be reduced to $u_4=unity$**
  - **Reduce the number of unit variables and constraints to improve performance of Gaussian Elimination.**

Osprey's Internal Structure



Courtesy ospreywatch.co.uk

Osprey at Work



Courtesy ospreywatch.co.uk

$$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL'_{rad}\}$$

```
extern $unity double alpha, N_A; extern $m double r_e;
typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 =     radiationLength(material);
    return  initEnergy / exp( thick           / X_0 );
}
```

$$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL'_{rad}\}$$

```
extern $unity double alpha, N_A; extern $m double r_e;
typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 =     radiationLength(material);
    return  initEnergy / exp( thick           / X_0 );
}
```

$$\frac{1}{X_0} = 4\alpha r_e^2 \frac{N_A}{A}\{Z^2[L_{rad} - f(Z)] + ZL'_{rad}\}$$

```
extern $unity double alpha, N_A; extern $m double r_e;
typedef struct {
    $kg double atomicWeight;
    $unity double atomicNumber;
} Element;

double radiationLength(Element * material) {
    double A = material->atomicWeight;
    double Z = material->atomicNumber;
    double L_rad = log( 184.15 / pow(Z, 1./3) );
    double Lp_rad = log( 1194. / pow(Z, 2./3) );

    return  ( 4 * alpha * r_e * r_e )
          * ( N_A / A ) * ( Z * Z * L_rad + Z * Lp_rad );
}

double finalEnergy(Element * material, $kg1m-3 double density,
                   $m double thick, $kg1m2s-2 double initEnergy){
    double X_0 = 1 / radiationLength(material);
    return  initEnergy / exp( thick * density / X_0 );
}
```

Ch® Mechanism Toolkit

- **Commercial libraries for computational mechanics**

Real Errors Found (1)

```
double couplerPointPos(double, $radian double theta, ……);
 ……

theta = linkLength / ( 1+lamda );
 ……
couplerPointPos(couplerLink, theta, ……);
```

- Developers tested the code, but inadequately.

Real Errors Found (2)

```
double angularAccel(double theta2,
    $radian1second-1 double omega2, $radian double theta3,
    double omega3, double alpha2);
 ……

int forceTorques( …… ) {
     ……
    angularAccel(theta2, theta3, omega2, omega3, alpha2);
     ……
}
```

- Arguments are passed in a wrong order in the function call.

Usability and Scalability

Lightweight Annotations

Scalable Performance

Number of Errors discovered

	Lines of Codes	Anno-tation Burden	Time Cost (s)	Peak Mem Cost (MB)	Peak Number of Constraints Before U/F	After U/F	Real Errors	False Posi-tives
Adopted from C-UNITS								
ex18.c	17	6 / 62	0.002	77.8	50	0	1	0
Adopted from SIUNITS								
fe.c	23	12 / 107	0.008	77.8	156	0	2	0
Adopted from Coil02								
coil.c	398	12 / 1492	0.046	78.0	859	15	0	3
Adopted from Ch©Mechanism Toolkit								
crankslider	1071	105 / 3299	0.135	78.5	2424	2	1	4
gearedfivebar	1120	62 / 2234	0.147	78.7	2720	26	2	0
fourbar	3166	264 / 10021	0.536	82.0	10741	63	0	3
sixbars	6564	331 / 13762	1.209	86.1	21772	168	0	4
Artificial Programs for Stress Testing								
big0.c	2705	0	0.444	81.8	10510	0	0	0
big1.c	11705	2 / 63716	2.536	93.3	39009	0	0	0
big2.c	96611	0	47.7	460.4	322027	0	0	0
big3.c	449384	1 / 2446636	968.2	1990.3	1497939	0	0	0
big4.c	122890	0	failed	failed	614411	135169	0	0

Effective U/F

Conclusion

- Osprey: a practical type system for unit correctness
  - Sound: does not miss unit errors
  - Usable: requires few unit annotations
  - Scalable: analyzes million line programs
- Future work
  - Sparse linear solvers for even better performance and scalability
  - Experiments on larger scale scientific programs

Thank you!

Question?

jiangl@cs.ucdavis.edu