ALGEBRA IN MY BROWSER AND RESPONSIBLE NUTRITION

Brian Beckman

November 2012

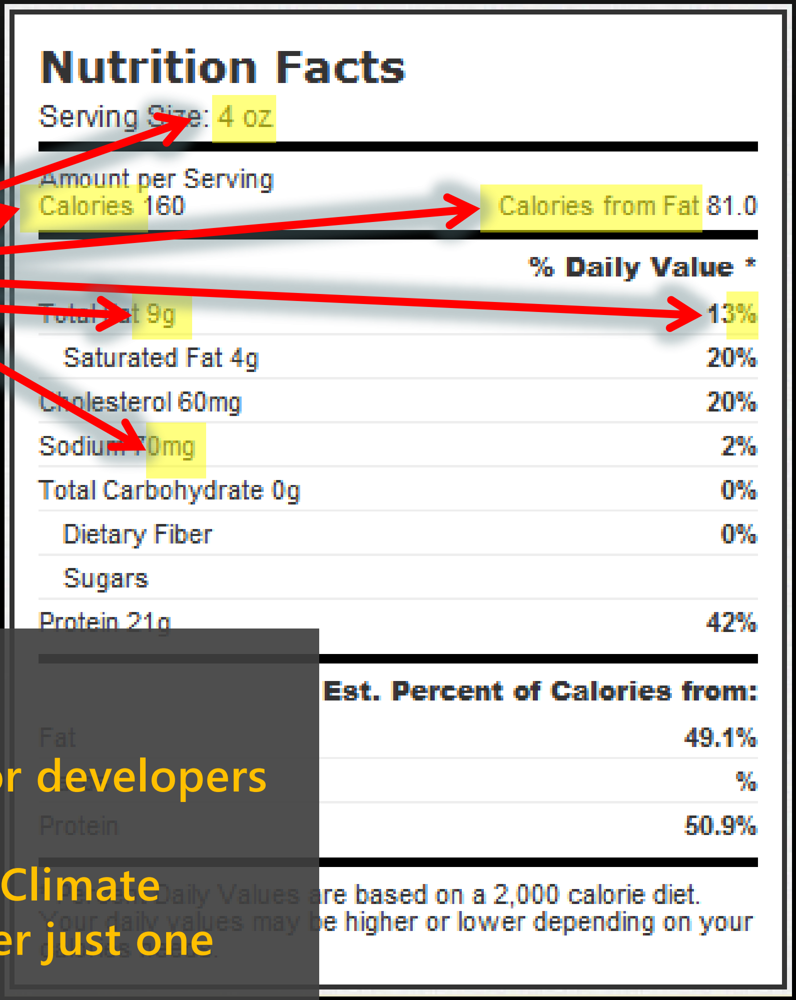
ALGEBRA IN MY BROWSER AND RESPONSIBLE ~~NUTRITION~~ DRINKING

Brian Beckman

November 2012

PICK A NUTRITION-FACTS LABEL AT RANDOM FROM THE WEB

There are at least eight different units of measure expressed or implied



Nutrition Facts

Serving Size: 4 oz

Amount per Serving

Calories 160 Calories from Fat 81.0

	% Daily Value *
Total Fat 9g	13%
Saturated Fat 4g	20%
Cholesterol 60mg	20%
Sodium 70mg	2%
Total Carbohydrate 0g	0%
Dietary Fiber	0%
Sugars	
Protein 21g	42%

	Est. Percent of Calories from:
Fat	49.1%
Carbs	%
Protein	50.9%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs.

What are the chances that the label is consistent: that implied values match stated values?

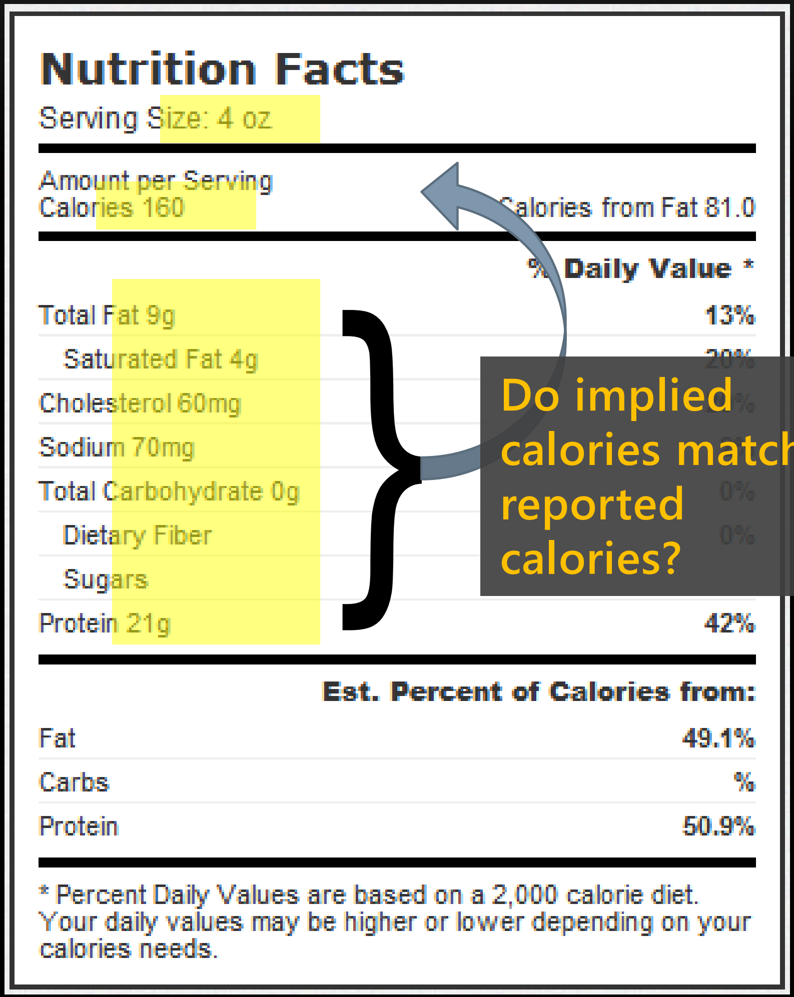
UNITS OF MEASURE

Notoriously tricky for developers

NASA crashed Mars Climate Observer in 1999 over just one units error

PICK A NUTRITION-FACTS LABEL AT RANDOM FROM THE WEB

Two questions we can ask immediately:



Nutrition Facts

Serving Size: 4 oz

Amount per Serving
Calories 160 Calories from Fat 81.0

	% Daily Value *
Total Fat 9g	13%
Saturated Fat 4g	20%
Cholesterol 60mg	20%
Sodium 70mg	3%
Total Carbohydrate 0g	0%
Dietary Fiber	0%
Sugars	
Protein 21g	42%

	Est. Percent of Calories from:
Fat	49.1%
Carbs	%
Protein	50.9%

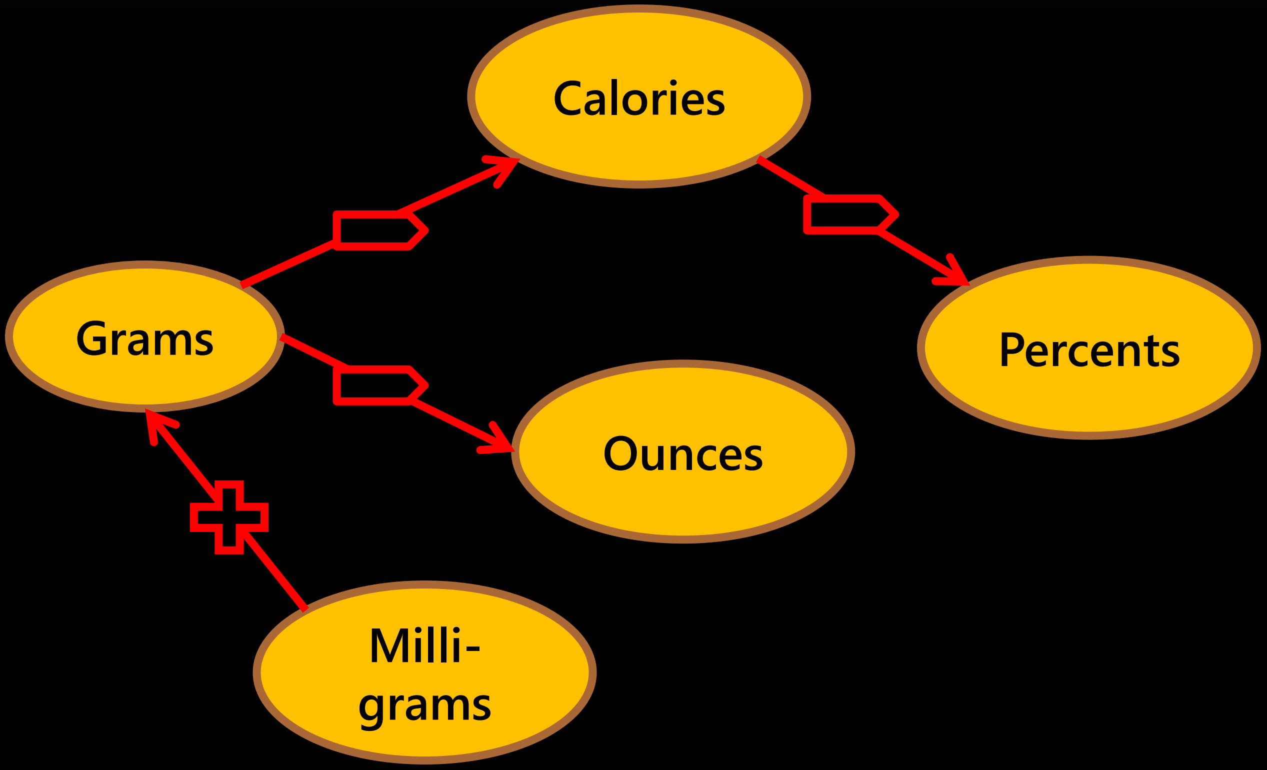
* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

Do these weights add up to this serving size?

Do implied calories match reported calories?

CENTRAL PROBLEM

How to write the validator app?



CENTRAL PROBLEM

JavaScript or C# programmer stuck with no innate way to track units of measure

```
var burgerNutritionFacts =
{ ServingSize          :   4  , // Ounce
  AmountPerServing     : 160  , // Calorie
  CaloriesFromFat      :  81.0, // Calorie
  SaturatedFat         :   4  , // Gram
  Cholesterol          :  60  , // Milligram
  Sodium               :  70  , // Milligram
  DietaryFiber         :   0  , // Gram
  Sugars               :   0  , // Gram
  TotalFat             :   9  , // Gram
  Protein              :  21  , // Gram
  TotalCarbohydrate    :   0  , // Gram
};
```

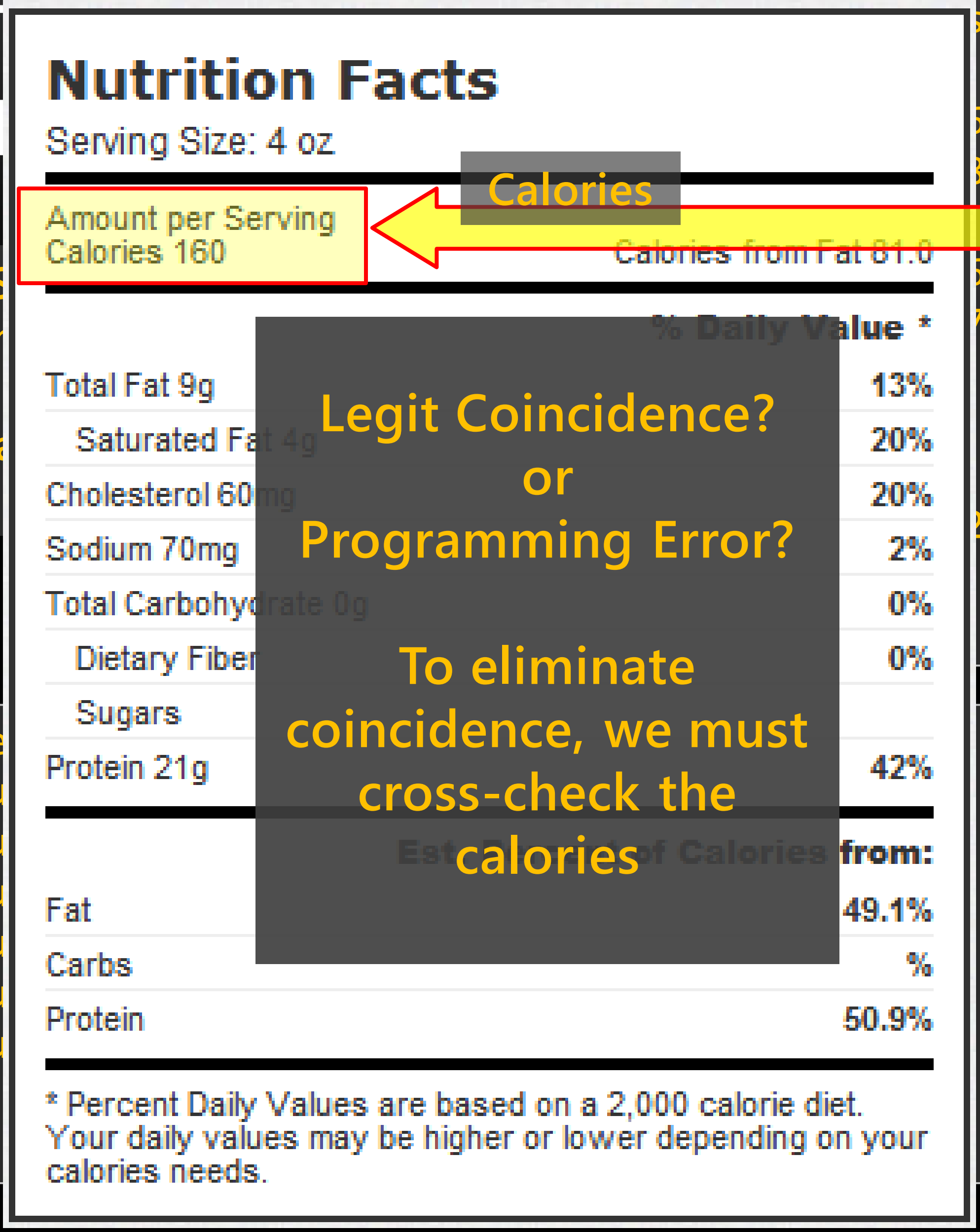
```
var addWeights = function(nutritionFacts) {,
return nutritionFacts.TotalFat + ,
       nutritionFacts.DietaryFiber +,
       nutritionFacts.Protein +,
       nutritionFacts.Cholesterol +,
       nutritionFacts.Sodium +,
       nutritionFacts.TotalCarbohydrate;
};
document.writeln(addWeights(burgerNutritionFacts));
```

```
160
```

Can you spot the mistake?

CENT
PRO

JavaS
progra
with no
to tra



Nutrition Facts

Serving Size: 4 oz

Amount per Serving
Calories 160

Calories from Fat 81.0

	% Daily Value *
Total Fat 9g	13%
Saturated Fat 4g	20%
Cholesterol 60mg	20%
Sodium 70mg	2%
Total Carbohydrate 0g	0%
Dietary Fiber	0%
Sugars	
Protein 21g	42%

Est.	of Calories from:
Fat	49.1%
Carbs	%
Protein	50.9%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

Calories

Legit Coincidence?
or
Programming Error?

To eliminate coincidence, we must cross-check the calories

```
s =
4   , // Ounce
50  , // Calorie
31.0, // Calorie
4   , // Gram
50  , // Milligram
70  , // Milligram
0   , // Gram
0   , // Gram
9   , // Gram
21  , // Gram
0   , // Gram
```

Is this another mistake?

weight

```
var addWe
return nu
       nu
       nu
       nu
       nu
       nu
};
document.
```

```
160
```

Can you spot the mistake?

CENTRAL PROBLEM

```
var burgerNutritionFacts =
{ ServingSize        : [  4  , "Ounce"     ],
  AmountPerServing   : [160  , "Calorie"   ],
  CaloriesFromFat    : [ 81.0, "Calorie"   ],
  SaturatedFat       : [  4  , "Gram"      ],
  Cholesterol        : [ 60  , "Milligram" ],
  Sodium             : [ 70  , "Milligram" ],
  DietaryFiber       : [  0  , "Gram"      ],
  Sugars             : [  0  , "Gram"      ],
  TotalFat           : [  9  , "Gram"      ],
  Protein            : [ 21  , "Gram"      ],
  TotalCarbohydrate  : [  0  , "Gram"      ]
};
```

TRY REFACTORING:
Invent a representation for quantities with units and write code to manage them

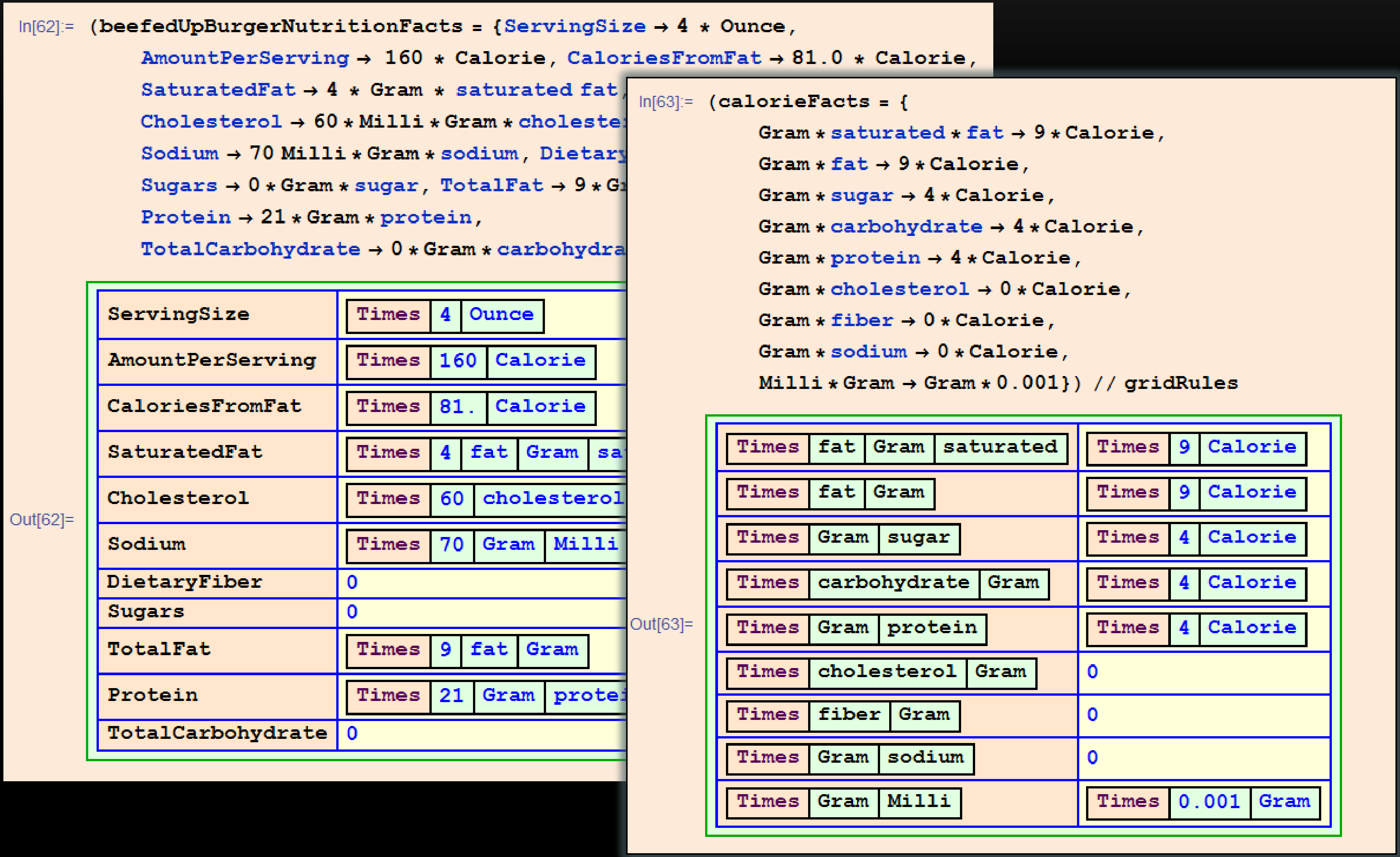
```
label.forEach(function (k, v) {
  switch(v[1]) {
    case "Ounce":
      result.k = [
        v[0] * 28.34952,
        "Gram"
      ],  ... more cases ...
  }  ... more code ...
});
```

We may write all the conversions we need for this app this way,

but generalize just a little, and we've invented an

ad-hoc, buggy, fragmentary simulation of computer algebra

OK SO WHY NOT JUST *USE* COMPUTER ALGEBRA? MATHEMATICA == BEST OF BREED



```
In[62]:= (beefedUpBurgerNutritionFacts = {ServingSize → 4 * Ounce,
           AmountPerServing → 160 * Calorie, CaloriesFromFat → 81.0 * Calorie,
           SaturatedFat → 4 * Gram * saturated fat,
           Cholesterol → 60 * Milli * Gram * choleste
           Sodium → 70 Milli * Gram * sodium, Dietary
           Sugars → 0 * Gram * sugar, TotalFat → 9 * Gr
           Protein → 21 * Gram * protein,
           TotalCarbohydrate → 0 * Gram * carbohydra

Out[62]=
   ServingSize         | Times | 4   | Ounce
   AmountPerServing    | Times | 160 | Calorie
   CaloriesFromFat     | Times | 81. | Calorie
   SaturatedFat        | Times | 4   | fat   | Gram | sa
   Cholesterol         | Times | 60  | cholesterol
   Sodium              | Times | 70  | Gram  | Milli
   DietaryFiber        | 0
   Sugars              | 0
   TotalFat            | Times | 9   | fat   | Gram
   Protein             | Times | 21  | Gram  | protei
   TotalCarbohydrate   | 0
```

```
In[63]:= (calorieFacts = {
           Gram * saturated * fat → 9 * Calorie,
           Gram * fat → 9 * Calorie,
           Gram * sugar → 4 * Calorie,
           Gram * carbohydrate → 4 * Calorie,
           Gram * protein → 4 * Calorie,
           Gram * cholesterol → 0 * Calorie,
           Gram * fiber → 0 * Calorie,
           Gram * sodium → 0 * Calorie,
           Milli * Gram → Gram * 0.001}) // gridRules

Out[63]=
   Times | fat | Gram | saturated      Times | 9 | Calorie
   Times | fat | Gram                  Times | 9 | Calorie
   Times | Gram | sugar                Times | 4 | Calorie
   Times | carbohydrate | Gram         Times | 4 | Calorie
   Times | Gram | protein              Times | 4 | Calorie
   Times | cholesterol | Gram          0
   Times | fiber | Gram                0
   Times | Gram | sodium               0
   Times | Gram | Milli                Times | 0.001 | Gram
```

NOW IMAGINE EVALUATING MATHEMATICA EXPRS IN JAVASCRIPT

Observe : syntactically, Mathematica is not horribly different from JavaScript

Observe : semantically, we

Mathematica's basic evalu

Observe : natural correspo

Mathematica expressions

Author in Mathematica and ru

Store expressions themselves

Avoid the trap of innovating ir

jump in, start swimming…

Language innovation is a risky, expensive socialogical process, not a technological process:

A CS intern can invent a new language, but getting adoption takes an

institution (e.g. ECMA),
an *authority* (e.g. Odersky),
or *luck* (e.g. CoffeeScript)

CHECKING WEIGHTS

Let arithmetic include symbolic constants

Represent objects and functions as lists of replacement rules

```
burgerNutritionFacts = {
  ServingSize        ->   4   * Ounce,
  AmountPerServing   -> 160   * Calorie,
  CaloriesFromFat    ->  81.0 * Calorie,
  SaturatedFat       ->   4   * Gram,
  Cholesterol        ->  60   * Milli * Gram,
  Sodium             ->  70   * Milli * Gram,
  DietaryFiber       ->   0   * Gram,
  Sugars             ->   0   * Gram,
  TotalFat           ->   9   * Gram,
  Protein            ->  21   * Gram,
  TotalCarbohydrate  ->   0   * Gram
}
```

OBJECTS ≅ FUNCTIONS ≅ RULES

```
var obj = {a: 1, b: 2};
```

```
obj.a ⇝ 1      obj["a"] ⇝ 1
```

Symbols in JavaScript
are actually strings!
∃ only before colons and after dots

```
function obj (x) {
   return x === "a" ? 1 :
         (x === "b" ? 2 : undefined);
}
```

```
obj("a") ⇝ 1
```

```
obj = {"a" -> 1, "b" -> 2, _ -> undefined}
```

```
"a" /. obj ⇝ 1
```

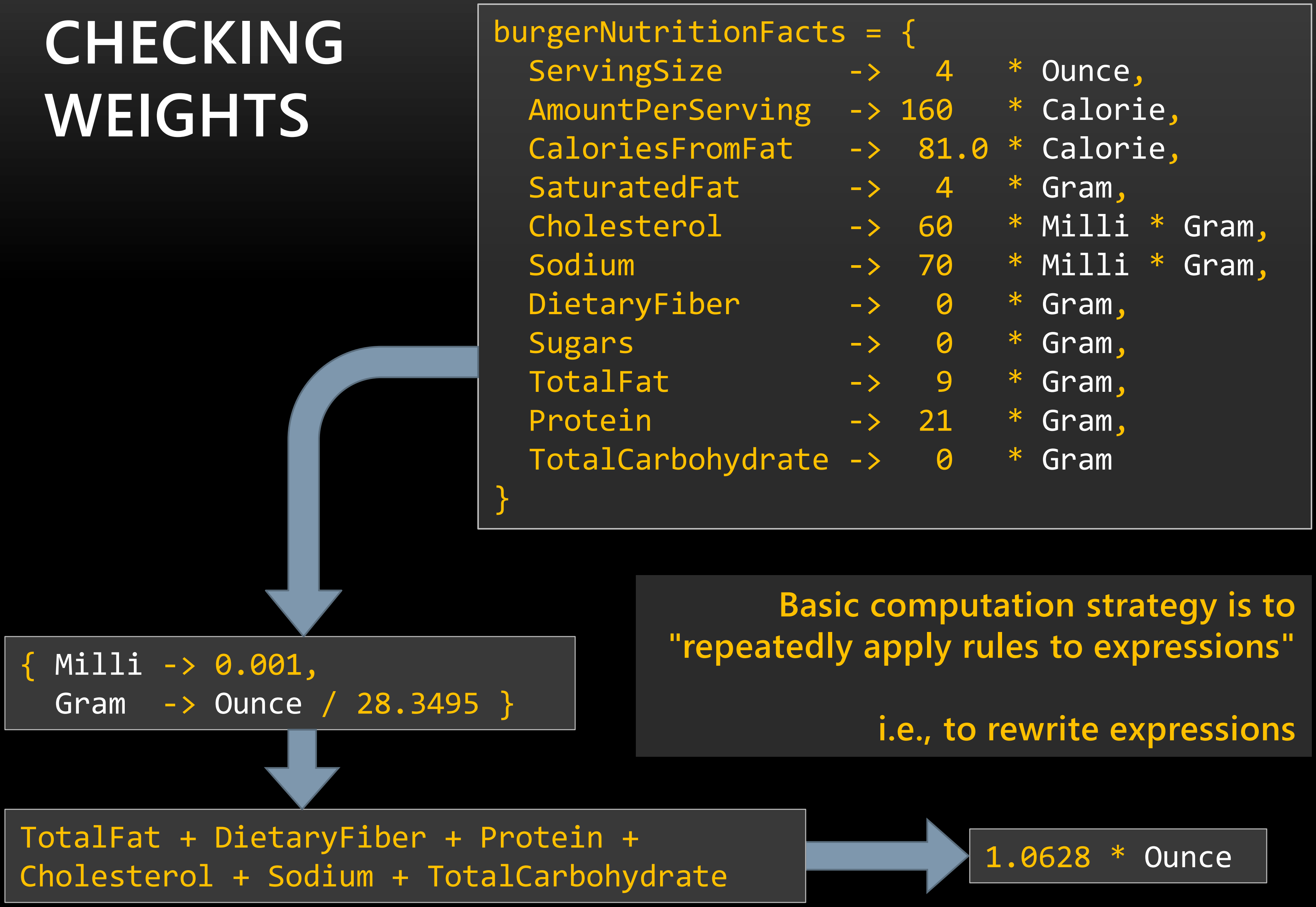
The meaning of expr /. rule is
"Apply the rule to the expression"

```
obj = {a -> 1, b -> 2, _ -> undefined}
```

```
a /. obj ⇝ 1
```

Real Computer Algebra has
symbolic constants as a distinct type

CHECKING WEIGHTS



```
burgerNutritionFacts = {
  ServingSize        ->   4   * Ounce,
  AmountPerServing   -> 160   * Calorie,
  CaloriesFromFat    ->  81.0 * Calorie,
  SaturatedFat       ->   4   * Gram,
  Cholesterol        ->  60   * Milli * Gram,
  Sodium             ->  70   * Milli * Gram,
  DietaryFiber       ->   0   * Gram,
  Sugars             ->   0   * Gram,
  TotalFat           ->   9   * Gram,
  Protein            ->  21   * Gram,
  TotalCarbohydrate  ->   0   * Gram
}
```

```
{ Milli -> 0.001,
  Gram  -> Ounce / 28.3495 }
```

```
TotalFat + DietaryFiber + Protein +
Cholesterol + Sodium + TotalCarbohydrate
```

```
1.0628 * Ounce
```

Basic computation strategy is to "repeatedly apply rules to expressions"

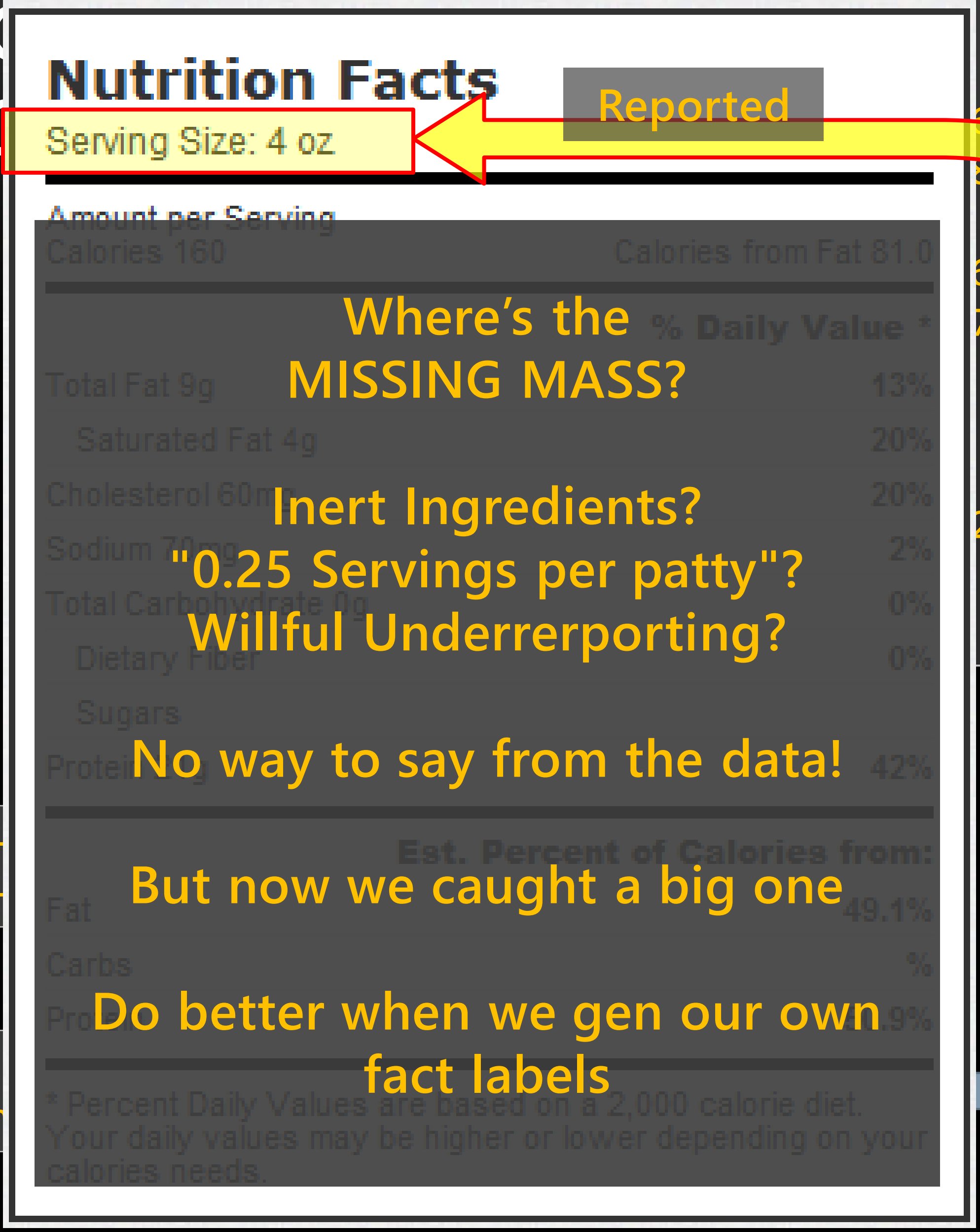
i.e., to rewrite expressions

CHEC
WEIG

```
4    * Ounce,
60   * Calorie,
81.0 * Calorie,
4    * Gram,
60   * Milli * Gram,
70   * Milli * Gram,
0    * Gram,
0    * Gram,
9    * Gram,
21   * Gram,
0    * Gram
```

```
{ Milli -
  Gram  -
```

```
TotalFat
Cholester
```



Nutrition Facts

Serving Size: 4 oz

Amount per Serving

Calories 160 Calories from Fat 81.0

% Daily Value *

Total Fat 9g 13%

Saturated Fat 4g 20%

Cholesterol 60mg 20%

Sodium 70mg 2%

Total Carbohydrate 0g 0%

Dietary Fiber 0%

Sugars

Protein 21g 42%

Est. Percent of Calories from:

Fat 49.1%

Carbs %

Protein 9%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

Where’s the MISSING MASS?

Inert Ingredients?
"0.25 Servings per patty"?
Willful Underrerporting?

No way to say from the data!

But now we caught a big one

Do better when we gen our own fact labels

Reported

Inferred

```
1.0628 * Ounce
```

TERM REWRITING

Term Rewriting is a general computational strategy

Can simulate lambda calculus and vice-versa

Commonplace in Computer Algebra and Theorem Proving

Enables very concise statements of sophisticated algorithms

Basic idea: “Replace patterns with expressions after variable substitution; iterate until nothing changes”

```
fib[n_ /; n <= 2] = 1
fib[n_] := fib[n - 1] + fib[n - 2]
fib[3] ~~> fib[2] + fib[1] ~~> 1 + 1 ~~> 2 ~~> 2 DONE!
```

Not a function call, but can simulate one

Can do much more, and much less, however

Patterns are like Regular Expressions, only they work on expressions themselves

Expressions are “auto-iconic” – they are their own representation language

CHECKING CALORIES

Not all grams are created equal –

grams fat ≠ grams carbs

```
beefedUpBurgerNutritionFacts = {
  ServingSize       ->   4 * Ounce,
  AmountPerServing  -> 160 * Calorie,
  CaloriesFromFat   ->  81 * Calorie,
  SaturatedFat      ->   4 * Gram * saturatedFat,
  Cholesterol       ->  60 * Milli * Gram * cholesterol,
  Sodium            ->  70 * Milli * Gram * sodium,
  DietaryFiber      ->   0 * Gram * fiber,
  Sugars            ->   0 * Gram * sugar,
  TotalFat          ->   9 * Gram * fat,
  Protein           ->  21 * Gram * protein,
  TotalCarbohydrate ->   0 * Gram * carbohydrate}
```

Now mine some more facts from the web

```
calorieFacts = {
  Gram * saturatedFat -> 9 * Calorie,
  Gram * fat          -> 9 * Calorie,
  Gram * sugar        -> 4 * Calorie,
  Gram * carbohydrate -> 4 * Calorie,
  Gram * protein      -> 4 * Calorie,
  Gram * cholesterol  -> 0 * Calorie,
  Gram * fiber        -> 0 * Calorie,
  Gram * sodium       -> 0 * Calorie,
  Milli * Gram        -> Gram * 0.001}}
```

We could dump all these zeros, but keep them around as reminders and for future generalization

CHECKING CALORIES

Rules are expressions too:

we can rewrite

rules-as-objects using rules-as-functions

```
beefedUpBurgerNutritionFacts = {
  ServingSize        ->   4 * Ounce,
  AmountPerServing   -> 160 * Calorie,
  CaloriesFromFat    ->  81 * Calorie,
  SaturatedFat       ->   4 * Gram * saturatedFat,
  Cholesterol        ->  60 * Milli * Gram * cholesterol,
  Sodium             ->  70 * Milli * Gram * sodium,
  DietaryFiber       ->   0 * Gram * fiber,
  Sugars             ->   0 * Gram * sugar,
  TotalFat           ->   9 * Gram * fat,
  Protein            ->  21 * Gram * protein,
  TotalCarbohydrate ->   0 * Gram * carbohydrate}
```

```
calorieFacts = {
  Gram * saturatedFat -> 9 * Calorie,
  Gram * fat          -> 9 * Calorie,
  Gram * sugar        -> 4 * Calorie,
  Gram * carbohydrate -> 4 * Calorie,
  Gram * protein      -> 4 * Calorie,
  Gram * cholesterol  -> 0 * Calorie,
  Gram * fiber        -> 0 * Calorie,
  Gram * sodium       -> 0 * Calorie,
  Milli * Gram        -> Gram * 0.001}}
```

```
{ ServingSize        ->   4 * Ounce,
  AmountPerServing   -> 160 * Calorie,
  CaloriesFromFat    ->  81 * Calorie,
  SaturatedFat       ->  36 * Calorie,
  Cholesterol        ->   0 * Calorie,
  Sodium             ->   0 * Calorie,
  DietaryFiber       ->   0 * Calorie,
  Sugars             ->   0 * Calorie,
  TotalFat           ->  81 * Calorie,
  Protein            ->  84 * Calorie,
  TotalCarbohydrate ->   0 * Calorie}
```

CHEC
CALO

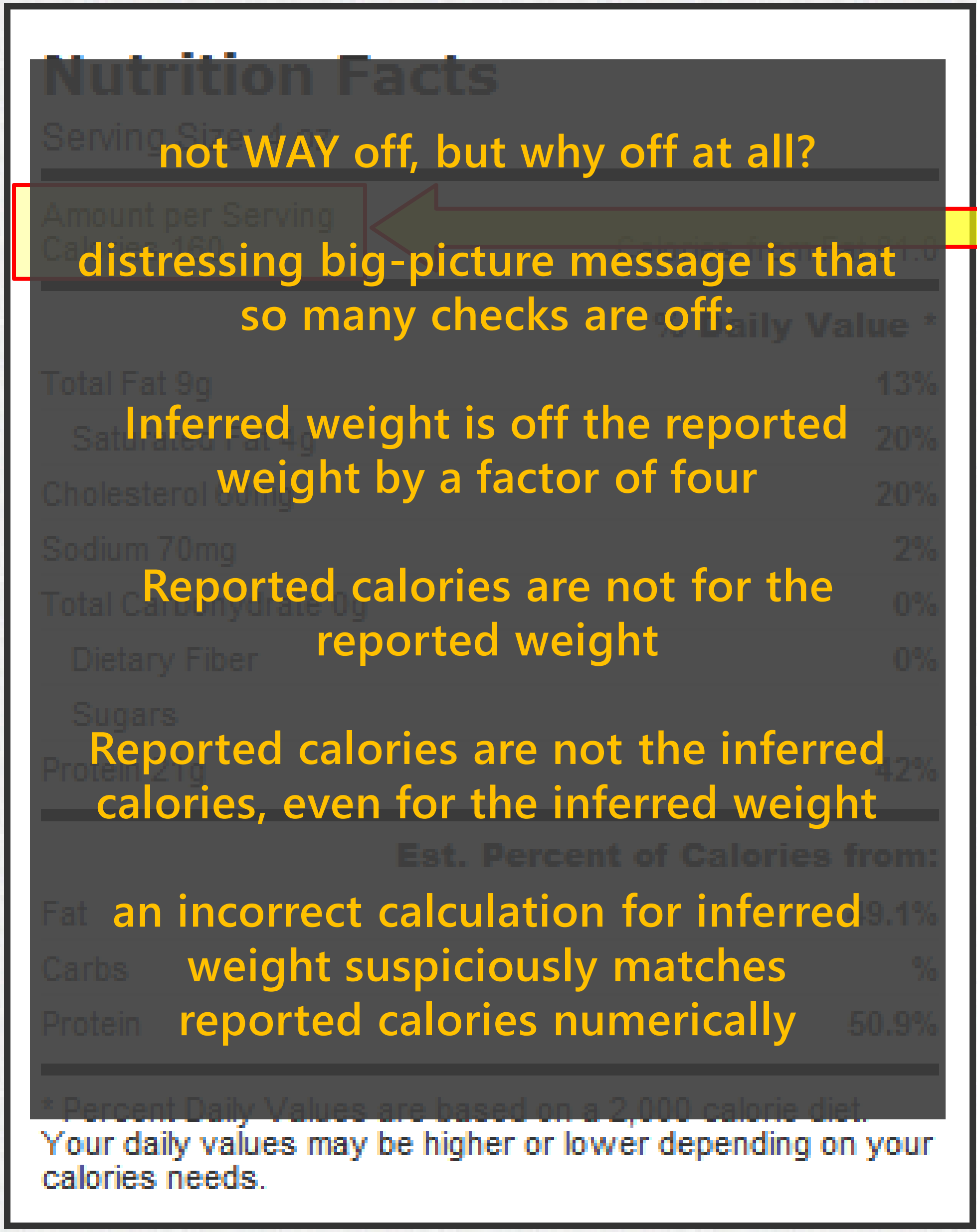
expres

we ca

rules-a

rules-as-

```
calorieFact
  Gram * sa
  Gram * fa
  Gram * su
  Gram * ca
  Gram * pr
  Gram * ch
  Gram * fi
  Gram * so
  Milli * G
```



Nutrition Facts

Serving Size 4 oz

Amount per Serving

Calories 160 Calories from Fat 81

% Daily Value *

Total Fat 9g 13%

Saturated Fat 4g 20%

Cholesterol 60mg 20%

Sodium 70mg 2%

Total Carbohydrate 0g 0%

Dietary Fiber 0%

Sugars

Protein 21g 42%

Est. Percent of Calories from:

Fat 9.1%

Carbs %

Protein 50.9%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

not WAY off, but why off at all?

distressing big-picture message is that so many checks are off:

Inferred weight is off the reported weight by a factor of four

Reported calories are not for the reported weight

Reported calories are not the inferred calories, even for the inferred weight

an incorrect calculation for inferred weight suspiciously matches reported calories numerically

```
{
nce,
lorie,
lorie,
am * saturatedFat,
lli * Gram * cholesterol,
lli * Gram * sodium,
am * fiber,
am * sugar,
am * fat,
am * protein,
am * carbohydrate}
```

```
e            ->   4 * Ounce,
erving       -> 160 * Calorie,
omFat        ->  81 * Calorie,
at           ->  36 * Calorie,
l            ->   0 * Calorie,
             ->   0 * Calorie,
er           ->   0 * Calorie,
             ->   0 * Calorie,
             ->  81 * Calorie,
             ->  84 * Calorie,
hydrate ->  165 * Calorie
```

SYMBOLIC ARITHMETIC JUSTIFIED

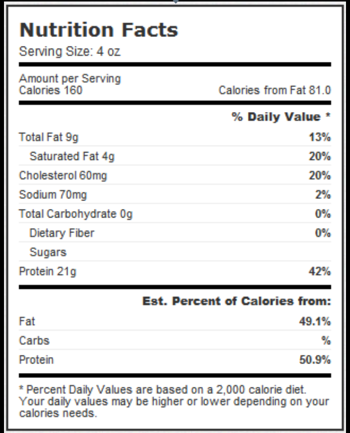
Robustly track and convert units of measure

"Easy on the Eyes": Minimal syntactic disruption to familiar JavaScript programming patterns

"Easy to Implement": Basic pattern-matching and rewriting is a small JavaScript library

Patterns-and-Rules strategy will extend our reach: next is robust creation of new labels from recipes...

HOW ABOUT NEW LABELS ON-THE-FLY?



Nutrition Facts

Serving Size: 4 oz

Amount per Serving

Calories 160	Calories from Fat 81.0

	% Daily Value *
Total Fat 9g	**13%**
Saturated Fat 4g	**20%**
Cholesterol 60mg	**20%**
Sodium 70mg	**2%**
Total Carbohydrate 0g	**0%**
Dietary Fiber	**0%**
Sugars	
Protein 21g	**42%**

	Est. Percent of Calories from:
Fat	**49.1%**
Carbs	**%**
Protein	**50.9%**

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

What would the label be for this meal?

STEPS IN THE SOLUTION

Ok, we are NOT doing image reco!

Assume we start with recipe data

Ingredients

2 cups (1-inch) diagonally sliced asparagus $

2 cups sliced carrot $

1 1/2 cups snow peas

1 tablespoon olive oil $

1/2 cup chopped onion $

2 garlic cloves, minced

6 cups hot cooked fettuccine (about 10 ounces uncooked pasta) $

1/2 cup (2 ounces) grated fresh Parmesan cheese, divided $

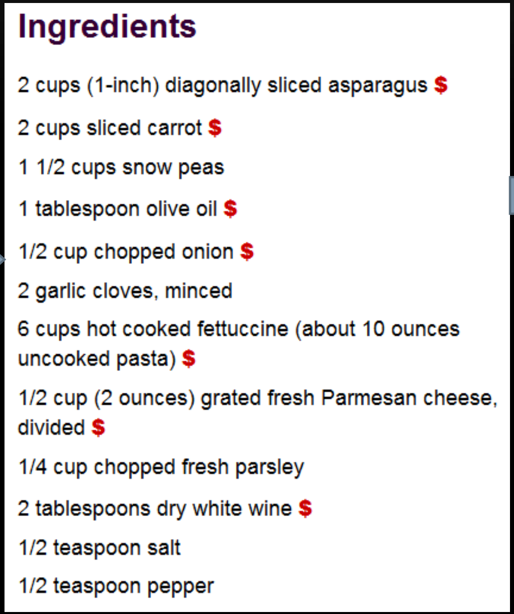
1/4 cup chopped fresh parsley

2 tablespoons dry white wine $

1/2 teaspoon salt

1/2 teaspoon pepper

STEPS IN THE SOLUTION



Ingredients

2 cups (1-inch) diagonally sliced asparagus $

2 cups sliced carrot $

1 1/2 cups snow peas

1 tablespoon olive oil $

1/2 cup chopped onion $

2 garlic cloves, minced

6 cups hot cooked fettuccine (about 10 ounces uncooked pasta) $

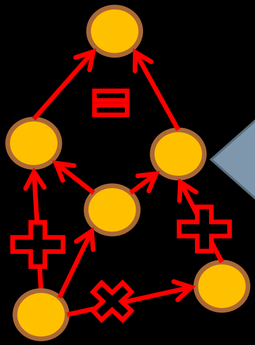
1/2 cup (2 ounces) grated fresh Parmesan cheese, divided $

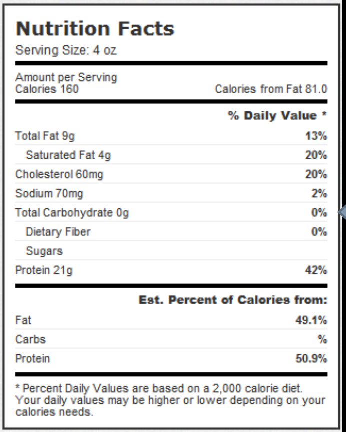
1/4 cup chopped fresh parsley

2 tablespoons dry white wine $

1/2 teaspoon salt

1/2 teaspoon pepper





Nutrition Facts

Serving Size: 4 oz

Amount per Serving Calories 160	Calories from Fat 81.0
	% Daily Value *
Total Fat 9g	13%
Saturated Fat 4g	20%
Cholesterol 60mg	20%
Sodium 70mg	2%
Total Carbohydrate 0g	0%
Dietary Fiber	0%
Sugars	
Protein 21g	42%
	Est. Percent of Calories from:
Fat	49.1%
Carbs	%
Protein	50.9%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calories needs.

THIS IS A VECTOR-SPACE PROBLEM

Each ingredient is a basis vector in Nutrition-Fact-Label (NFL) Vector Space

- *Normalize* them into unit vectors

The recipe is a sum of scaled ingredients

- Must put all ingredients into commensurable units
- i.e., *Canonicalize* the units

COMPUTATIONAL PLAN

Rewrite the Recipe in grams

Convert volumes to masses via data-mined densities

Convert whole-item ingredients to masses via data-mined averages

Data-mine ingredients

Normalize each into a 1-gram “serving size”

Proportionately scale all dependent quantities (calories, percents)

Lookup, scale and sum to write new NFL

Optional postprocess: scale and convert result into user-specified units

REWRITING A RECIPE

write the recipe as a list of quantified ingredients

some volumes
some masses
some whole-items

It's ok to multiply numbers by symbols by strings – such has no other meaning than itself until there is a rule to transform it

```
myRecipe = {
  1.0  * Tablespoon * "olive oil",
 16.0  * Ounce      * "zucchini",
  3.5  * Teaspoon   * "salt",
  1.5  * Pound      * "eggplant",
  1.0               * "onion",
  2.0               * "bell pepper",
 14.5  * Ounce      * "stewed tomato",
  0.5  * Teaspoon   * "black pepper",
  0.5  * Teaspoon   * "dried basil",
  0.5  * Teaspoon   * "sugar",
 12.0  * Ounce      * "pasta",
  0.25 * Cup        * "parmesan cheese"}
```

Now mine density facts from the web

```
density["olive oil"]       = Mean[{6.68,7.67}] * Pound / Gallon;
density["salt"]            = 5.69 Gram / Teaspoon;
density["black pepper"]    = 2.1 Gram / Teaspoon;
density["dried basil"]     = 1.0 Gram / Teaspoon;
density["sugar"]           = 4.2 Gram / Teaspoon;
density["parmesan cheese"] = 88 Gram / Cup;
```

REWRITING A RECIPE

Bracket notation is just another way of writing a rule…
a rule that is always applied whenever seen…
a rule stored in session state – on purpose

Resembles function definition on purpose

left-hand side = pattern (like a regular expression)
right-hand side = replacement

```
density["olive oil"]       = Mean[{6.68,7.67}] * Pound / Gallon;
density["salt"]            = 5.69 Gram / Teaspoon;
density["black pepper"]    = 2.1 Gram / Teaspoon;
density["dried basil"]     = 1.0 Gram / Teaspoon;
density["sugar"]           = 4.2 Gram / Teaspoon;
density["parmesan cheese"] = 88 Gram / Cup;
```

REWRITING A RECIPE

Must create rules for converting quantified ingredients into weights in grams

```
weightRuleFromQuantifiedIngredientVolume[
  quantity_?NumberQ   *
  ingredient_String   *
  volume : (Teaspoon | Tablespoon | Cup | FluidOunce | Pint | Gallon)] :=
```

The pattern has *variables_* distinguished by underscores

They're bound to actual values during the match

This pattern will match any line in the recipe that is a numerical *quantity_* times an *ingredient_* of type *String* times one of the given symbolic constants; the matched one is given the name *volume*

The ?NumberQ test could be any other boolean-valued expression

REWRITING A RECIPE

```
weightRuleFromQuantifiedIngredientVolume[
  quantity_?NumberQ   *
  ingredient_String   *
  volume : (Teaspoon | Tablespoon | Cup | FluidOunce | Pint | Gallon)] :=

ingredient * volume ->
  ingredient * gramPerTargetVolumeFromDensity[
    volume,
    density[ingredient] ] * volume;
```

If we match such a line, rewrite it as a rule that, when applied, will convert the ingredient times the volume into the ingredient times the gram-density: grams per unit of the matched volume

REWRITING A RECIPE

```
weightRuleFromQuantifiedIngredientVolume[
  quantity_?NumberQ   *
  ingredient_String   *
  volume : (Teaspoon | Tablespoon | Cup | FluidOunce | Pint | Gallon)] :=

ingredient * volume ->
  ingredient * gramPerTargetVolumeFromDensity[
    volume,
    density[ingredient] ] * volume;
```

```
gramPerTargetVolumeFromDensity[
  targetVolume_,
  d_?NumberQ * weight_ / volume_] :=

 (d * Convert[weight, Gram]) / Convert[volume, targetVolume]
```

rewrite data-mined densities in terms of recipe volumes

REWRITING A RECIPE

```
weightRuleFromQuantifiedIngredientVolume[
  quantity_?NumberQ   *
  ingredient_String   *
  volume : (Teaspoon | Tablespoon | Cup | FluidOunce | Pint | Gallon)] :=

ingredient * volume ->
  ingredient * gramPerTargetVolumeFromDensity[
    volume,
    density[ingredient] ] * volume;
```

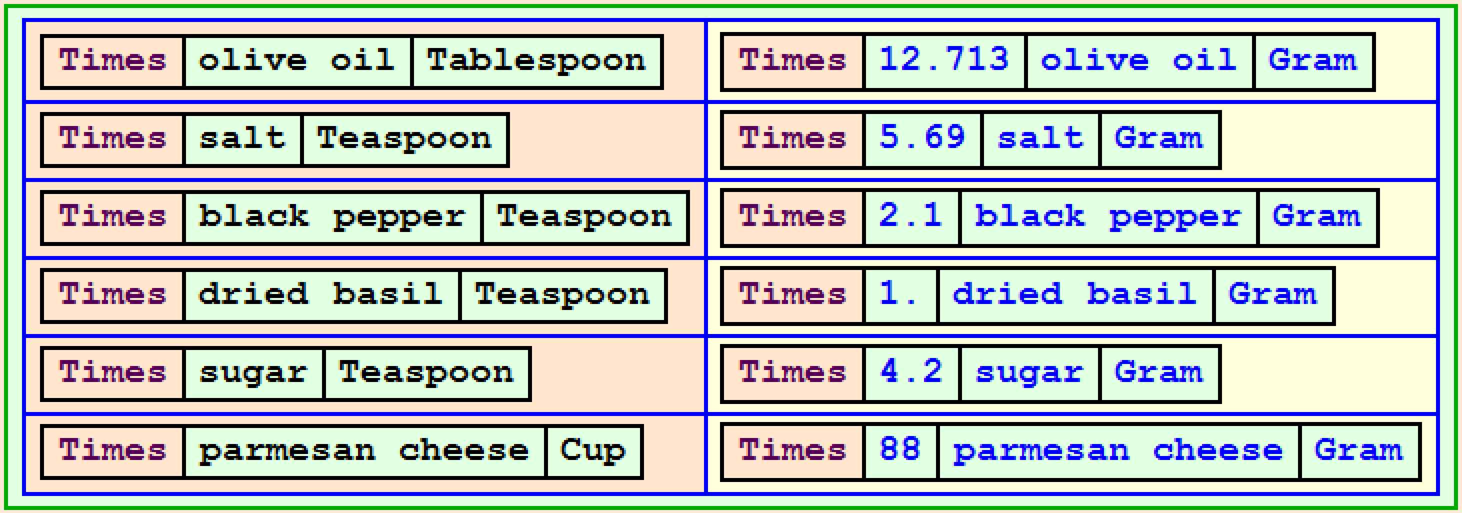
```
gramPerTargetVolumeFromDensity[
  targetVolume_,
  d_?NumberQ * weight_ / volume_] :=

 (d * Convert[weight, Gram]) / Convert[volume, targetVolume]
```

```
weightRuleFromQuantifiedIngredientVolume[___] := {}
```

```
volumeRules =
 SelectMany[myRecipe, weightRuleFromQuantifiedIngredientVolume]
```

Need one more rule to cover cases that don't match…
Then the magical monadic bind: SelectMany…
the Swiss army knife of programming with collections

REWRITING A RECIPE



Times olive oil Tablespoon	Times 12.713 olive oil Gram
Times salt Teaspoon	Times 5.69 salt Gram
Times black pepper Teaspoon	Times 2.1 black pepper Gram
Times dried basil Teaspoon	Times 1. dried basil Gram
Times sugar Teaspoon	Times 4.2 sugar Gram
Times parmesan cheese Cup	Times 88 parmesan cheese Gram

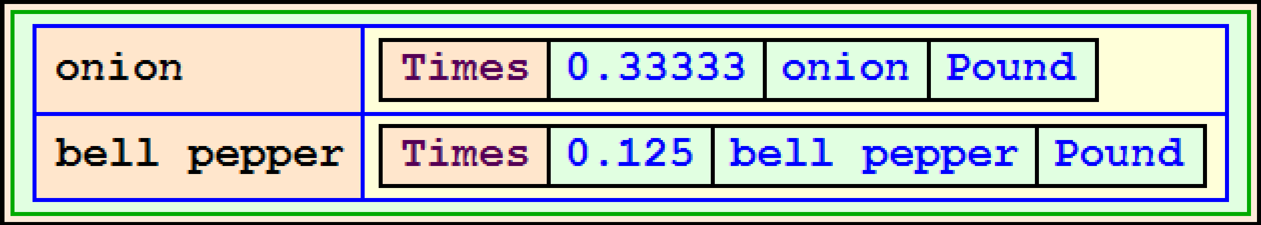
```
volumeRules =
 SelectMany[myRecipe, weightRuleFromQuantifiedIngredientVolume]
```

REWRITING A RECIPE

likewise for whole items:

```
wholeItemWeight["onion"] = (1.0/3) Pound;
wholeItemWeight["bell pepper"] = 0.5 Pound/4;
```

```
weightRuleFromQuantifiedWholeItemIngredient[_ * _String * _Symbol] = {};

weightRuleFromQuantifiedWholeItemIngredient[_?NumberQ * ingredient_] :=
  ingredient -> ingredient * wholeItemWeight[ingredient];

weightRuleFromQuantifiedWholeItemIngredient[___] = {};

wholeItemRules =
  SelectMany[myRecipe,
    weightRuleFromQuantifiedWholeItemIngredient]
```

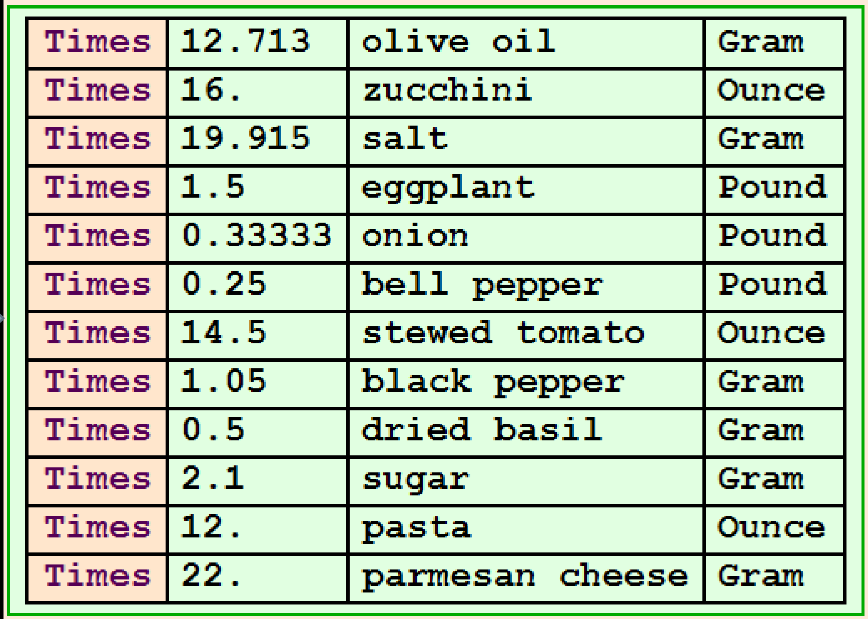


onion	Times | 0.33333 | onion | Pound
bell pepper	Times | 0.125 | bell pepper | Pound

REWRITING A RECIPE

```
myRecipe = {
  1.0  * Tablespoon * "olive oil",
 16.0  * Ounce      * "zucchini",
  3.5  * Teaspoon   * "salt",
  1.5  * Pound      * "eggplant",
  1.0               * "onion",
  2.0               * "bell pepper",
 14.5  * Ounce      * "stewed tomato",
  0.5  * Teaspoon   * "black pepper",
  0.5  * Teaspoon   * "dried basil",
  0.5  * Teaspoon   * "sugar",
 12.0  * Ounce      * "pasta",
  0.25 * Cup        * "parmesan cheese"}
```

```
myRecipe /.
volumeRules /.
wholeItemRules
```

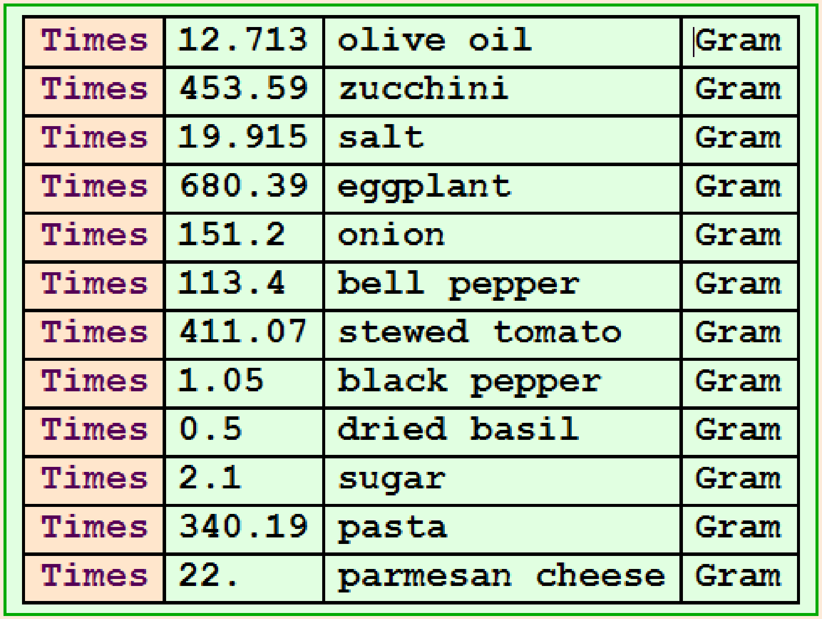


Times	12.713	olive oil	Gram
Times	16.	zucchini	Ounce
Times	19.915	salt	Gram
Times	1.5	eggplant	Pound
Times	0.33333	onion	Pound
Times	0.25	bell pepper	Pound
Times	14.5	stewed tomato	Ounce
Times	1.05	black pepper	Gram
Times	0.5	dried basil	Gram
Times	2.1	sugar	Gram
Times	12.	pasta	Ounce
Times	22.	parmesan cheese	Gram

REWRITING A RECIPE

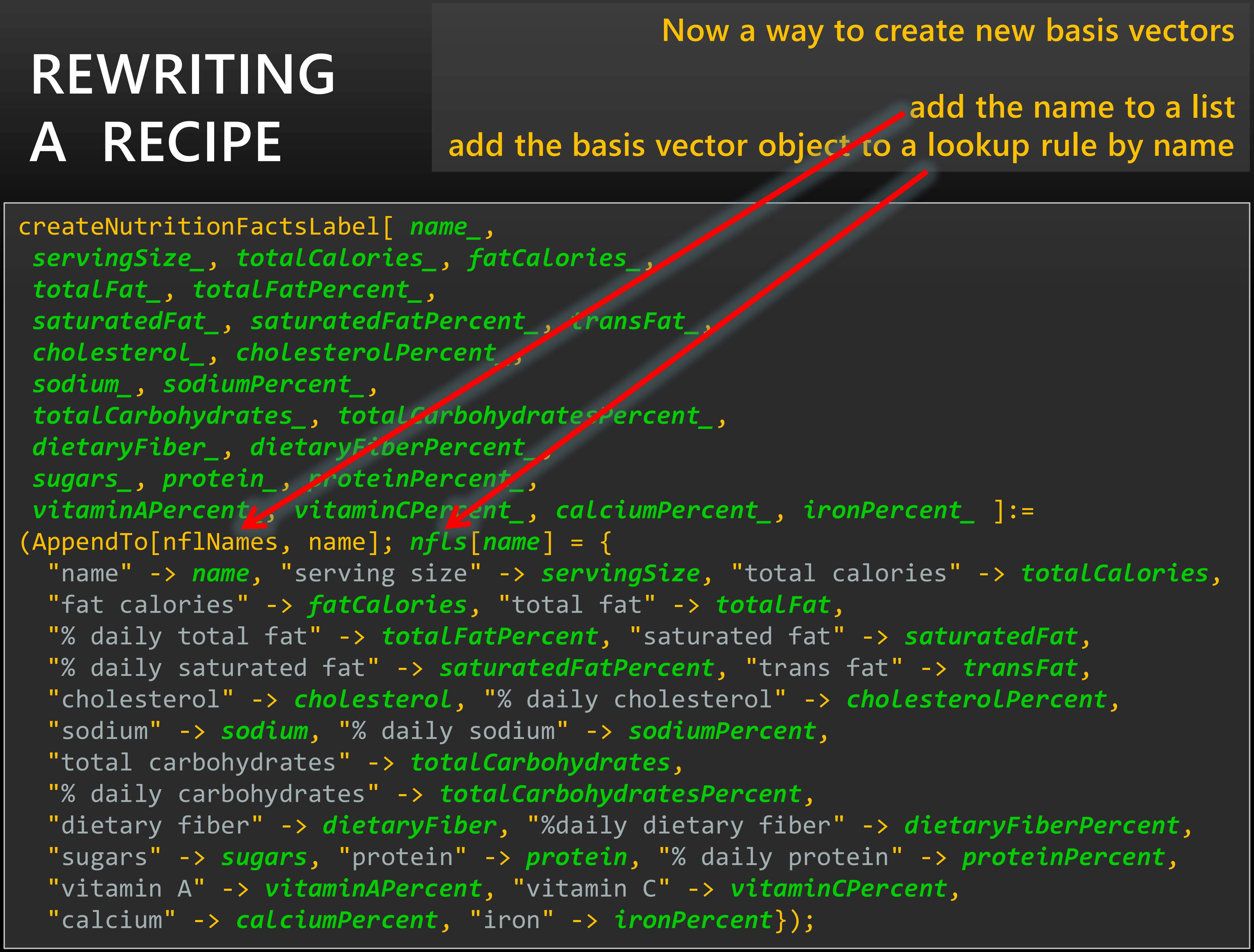
```
myRecipe = {
   1.0  * Tablespoon * "olive oil",
  16.0  * Ounce      * "zucchini",
   3.5  * Teaspoon   * "salt",
   1.5  * Pound      * "eggplant",
   1.0               * "onion",
   2.0               * "bell pepper",
  14.5  * Ounce      * "stewed tomato",
   0.5  * Teaspoon   * "black pepper",
   0.5  * Teaspoon   * "dried basil",
   0.5  * Teaspoon   * "sugar",
  12.0  * Ounce      * "pasta",
   0.25 * Cup        * "parmesan cheese"}
```

```
recipeInGrams =
 Map[
  Function[ingredient,
   Convert[ingredient, Gram]],
  myRecipe /.
    volumeRules /.
      wholeItemRules
```



Times	12.713	olive oil	Gram
Times	453.59	zucchini	Gram
Times	19.915	salt	Gram
Times	680.39	eggplant	Gram
Times	151.2	onion	Gram
Times	113.4	bell pepper	Gram
Times	411.07	stewed tomato	Gram
Times	1.05	black pepper	Gram
Times	0.5	dried basil	Gram
Times	2.1	sugar	Gram
Times	340.19	pasta	Gram
Times	22.	parmesan cheese	Gram

REWRITING A RECIPE



Now a way to create new basis vectors

add the name to a list

add the basis vector object to a lookup rule by name

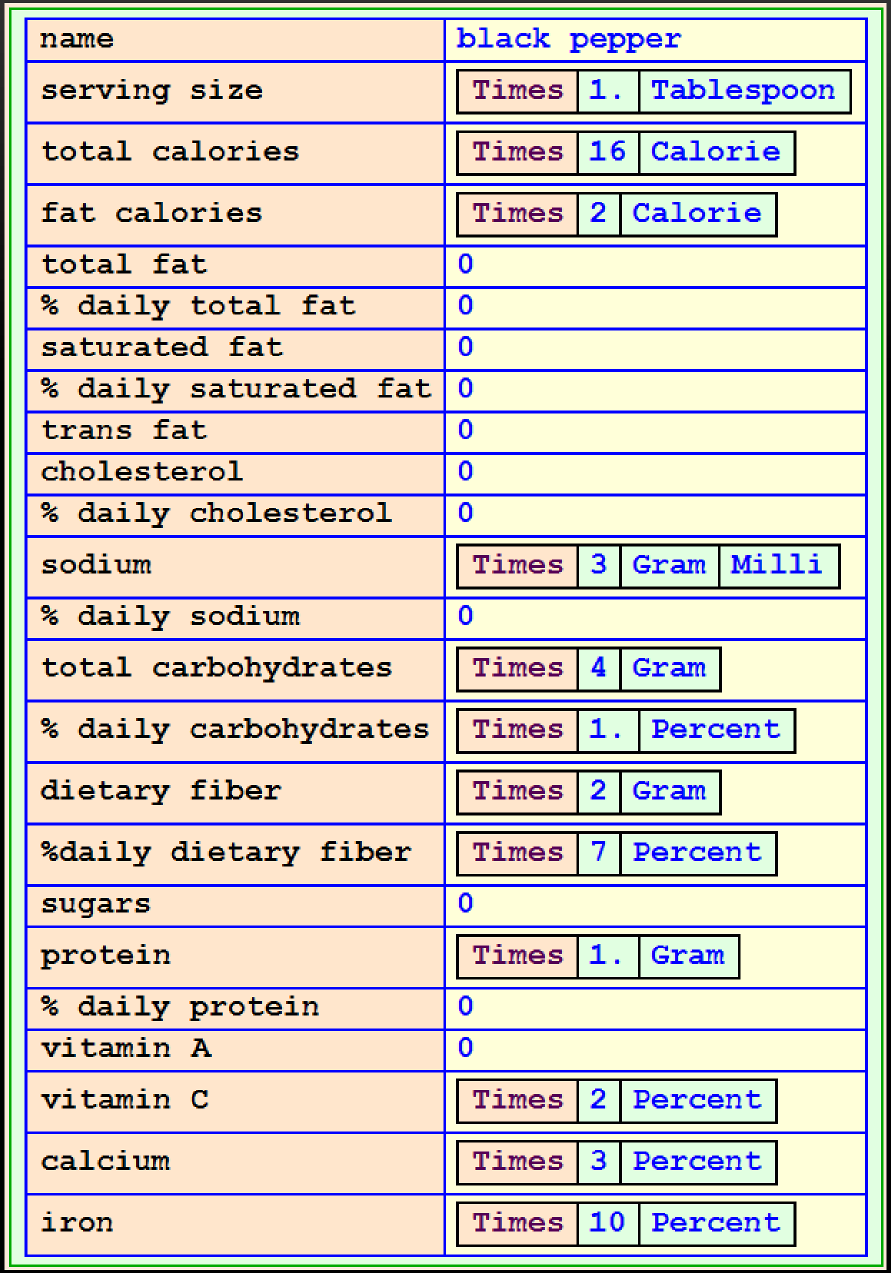
```
createNutritionFactsLabel[ name_,
 servingSize_, totalCalories_, fatCalories_,
 totalFat_, totalFatPercent_,
 saturatedFat_, saturatedFatPercent_, transFat_,
 cholesterol_, cholesterolPercent_,
 sodium_, sodiumPercent_,
 totalCarbohydrates_, totalCarbohydratesPercent_,
 dietaryFiber_, dietaryFiberPercent_,
 sugars_, protein_, proteinPercent_,
 vitaminAPercent_, vitaminCPercent_, calciumPercent_, ironPercent_ ]:=
(AppendTo[nflNames, name]; nfls[name] = {
  "name" -> name, "serving size" -> servingSize, "total calories" -> totalCalories,
  "fat calories" -> fatCalories, "total fat" -> totalFat,
  "% daily total fat" -> totalFatPercent, "saturated fat" -> saturatedFat,
  "% daily saturated fat" -> saturatedFatPercent, "trans fat" -> transFat,
  "cholesterol" -> cholesterol, "% daily cholesterol" -> cholesterolPercent,
  "sodium" -> sodium, "% daily sodium" -> sodiumPercent,
  "total carbohydrates" -> totalCarbohydrates,
  "% daily carbohydrates" -> totalCarbohydratesPercent,
  "dietary fiber" -> dietaryFiber, "%daily dietary fiber" -> dietaryFiberPercent,
  "sugars" -> sugars, "protein" -> protein, "% daily protein" -> proteinPercent,
  "vitamin A" -> vitaminAPercent, "vitamin C" -> vitaminCPercent,
  "calcium" -> calciumPercent, "iron" -> ironPercent});
```

REWRITING A RECIPE

```
createNutritionFactsLabel[
"black pepper", 1. Tablespoon,
16 Calorie, 2 Calorie, 0 Gram,
0 Percent, 0 Gram, 0 Percent, 0 Gram,
0 Gram, 0 Percent, 3 Milli Gram,
0 Percent, 4 Gram, 1. Percent,
2 Gram, 7 Percent, 0 Gram, 1. Gram,
0 Percent, 0 Percent, 2 Percent,
3 Percent, 10 Percent]
```

Terse and convenient for database

There are lots of ways we could make it prettier and more robust



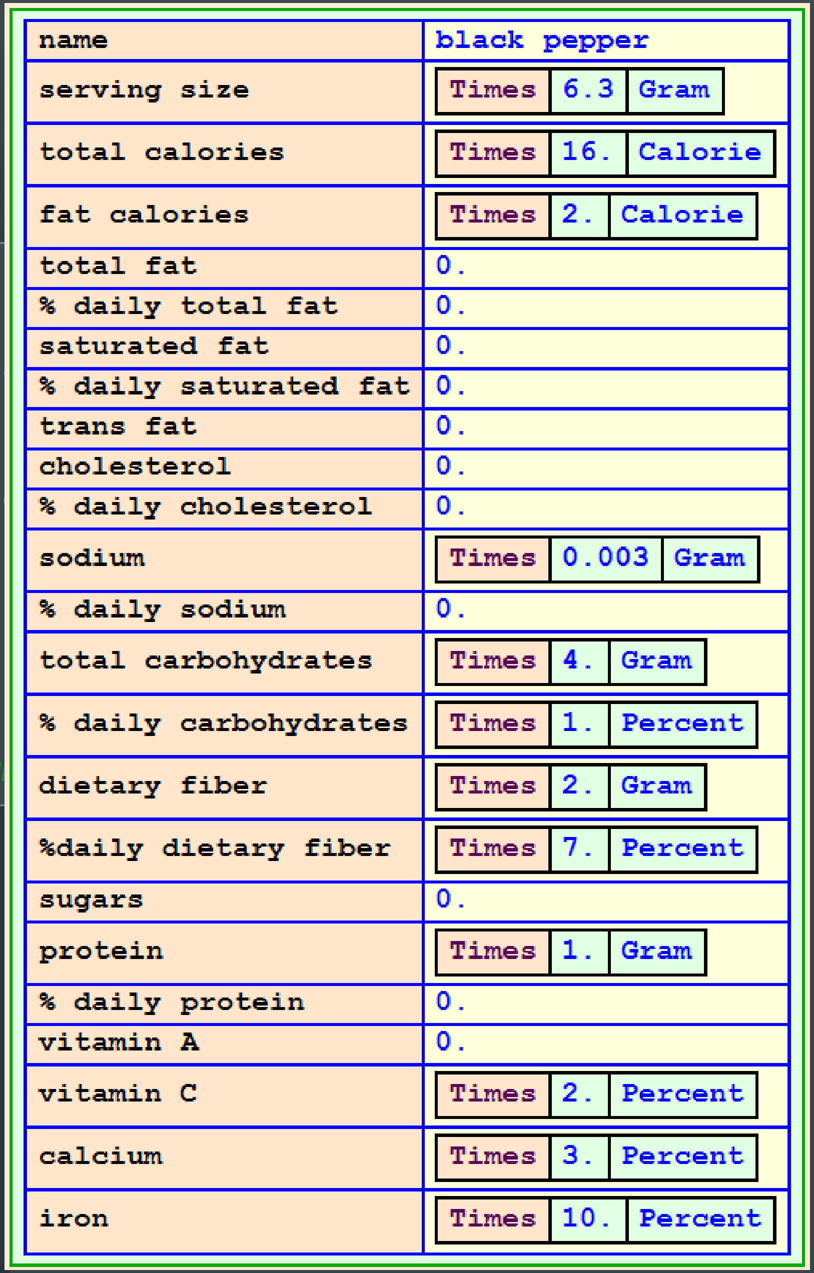
name	black pepper
serving size	Times 1. Tablespoon
total calories	Times 16 Calorie
fat calories	Times 2 Calorie
total fat	0
% daily total fat	0
saturated fat	0
% daily saturated fat	0
trans fat	0
cholesterol	0
% daily cholesterol	0
sodium	Times 3 Gram Milli
% daily sodium	0
total carbohydrates	Times 4 Gram
% daily carbohydrates	Times 1. Percent
dietary fiber	Times 2 Gram
%daily dietary fiber	Times 7 Percent
sugars	0
protein	Times 1. Gram
% daily protein	0
vitamin A	0
vitamin C	Times 2 Percent
calcium	Times 3 Percent
iron	Times 10 Percent

REWRITING A RECIPE

```
canonicalizeUnits[nfl_] := (* for each rule
  Map[ Function[ rule, rule[[1]] -> Convert
    (nfl /. {  (* pattern to match against
      (keyWithVolume_ ->  (* this arrow is
         amount_?NumberQ  *
         volume : (Teaspoon | Tablespoon |

      :>  (* this arrow is part of the rule

      (* this is the resulting new rule *)
      keyWithVolume -> amount  *  volume  *
        gramPerTargetVolumeFromDensity[volu
```

Convert all weights and volumes to grams

honor "hippocratic principle" – don't damage lines that don't match



name	black pepper
serving size	Times 6.3 Gram
total calories	Times 16. Calorie
fat calories	Times 2. Calorie
total fat	0.
% daily total fat	0.
saturated fat	0.
% daily saturated fat	0.
trans fat	0.
cholesterol	0.
% daily cholesterol	0.
sodium	Times 0.003 Gram
% daily sodium	0.
total carbohydrates	Times 4. Gram
% daily carbohydrates	Times 1. Percent
dietary fiber	Times 2. Gram
%daily dietary fiber	Times 7. Percent
sugars	0.
protein	Times 1. Gram
% daily protein	0.
vitamin A	0.
vitamin C	Times 2. Percent
calcium	Times 3. Percent
iron	Times 10. Percent

REWRITING A RECIPE

A list of norms, a list of basis vectors

A function to scale any vector

➔ normalized basis vectors

```
nflList = Map[ Function[name, nfls[name]], nflNames ];
(* make a list of nfls from the lookup rules, for mapping *)
canonicalizedNfls = Map[ canonicalizeUnits, nflList ];
```

```
norms = Map[ Function[nfl, ("serving size" / Gram /. nfl)], canonicalizedNfls ]

{216., 273.12, 82., 160., 186., 101., 6.3, 1., 2., 128., 100., 100.}
```

```
scaleNfl[nfl_, scalar_] :=
 Map[
  Function[line, If[line[[1]] === "name",
    line, (* skip the name line (hippocratically) *)
    line[[1]] -> line[[2]] * scalar]],
  nfl]
```

```
normalizedNfls = Zip[ scaleNfl, {canonicalizedNfls, 1 / norms} ];
```

```
normalizedNflsObj =
(* convert the list into an object (a list of rules) *)
  Map[ Function[nfl, ("name" /. nfl) -> nfl ], normalizedNfls];
```

REWRITING A RECIPE

A function to scale an NFL from an ingredient in grams

A function to add two NFLs

A Fold and an ad-hoc scale

```
scaledNflFromIngredient[qtty_?NumberQ * name_String * Gram] :=
 If[(name /. normalizedNflsObj) === name,
 (* ingredient wasn't in DB *) {}, (* for SelectMany to flatten *)
  {scaleNfl[name /. normalizedNflsObj, qtty]}]
```

```
scaledNfls = SelectMany[recipeInGrams, scaledNflFromIngredient]
```

```
sumNfls[nfl1_, nfl2_] :=
 Zip[
  Function[{line1, line2},
   If[line1[[1]] === line2[[1]],
    line1[[1]] -> (line1[[2]] + line2[[2]]),
    Throw["dimensions didn't match"]]],
  {nfl1, nfl2}]
```

```
scaleNfl[Fold[sumNfls, First[scaledNfls], Rest[scaledNfls]], 1/6]
```

REWRITING A RECIPE

cale an NFL from an ingredient in grams

ion to add two NFLs

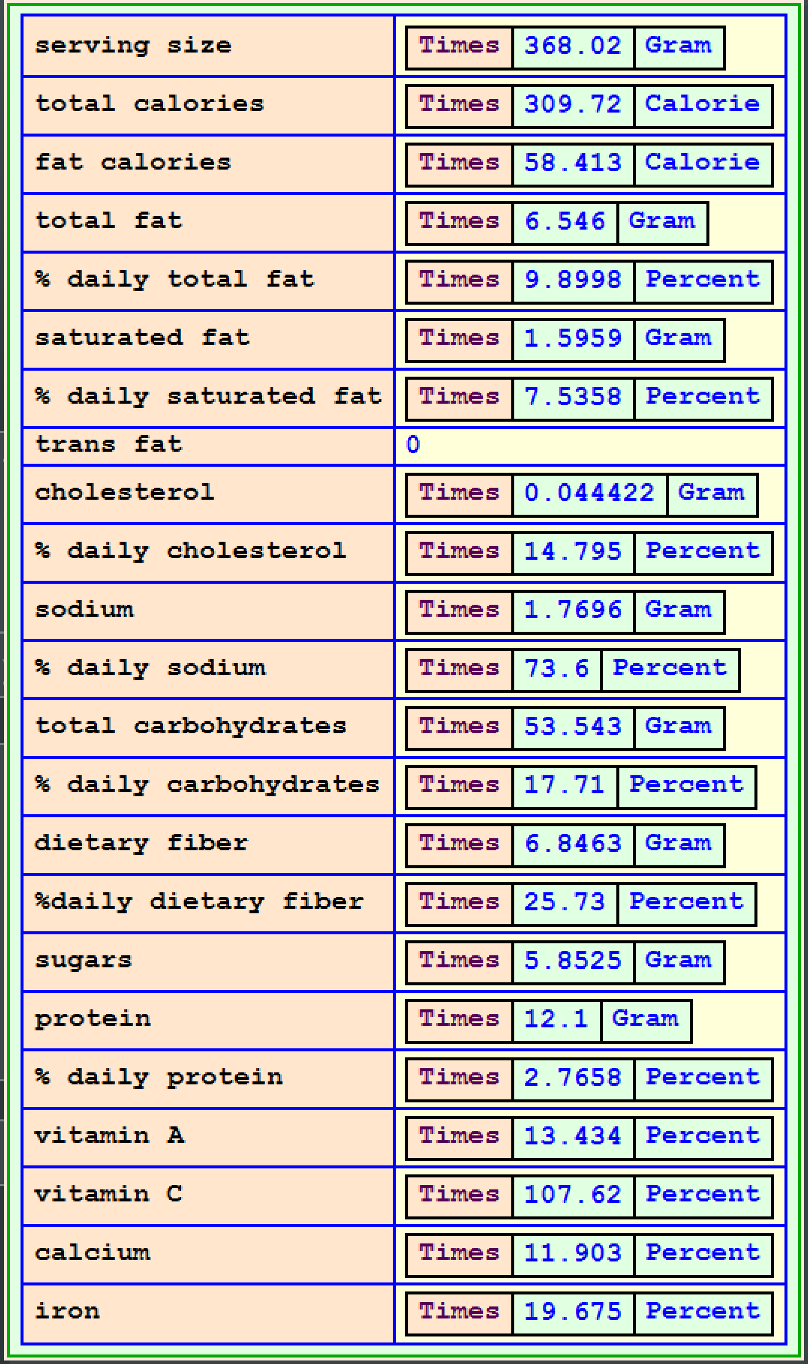
and an ad-hoc scale

```
scaledNflFromIngredien         =
 If[(name /. normalize
 (* ingredient wasn't          en *)
  {scaleNfl[name /. no
```

```
scaledNfls = SelectMan          t]
```

```
sumNfls[nfl1_, nfl2_]
 Zip[
  Function[{line1, lin
   If[line1[[1]] === l
    line1[[1]] -> (lin
    Throw["dimensions
  {nfl1, nfl2}]
```

```
scaleNfl[Fold[sumNfls,          1/6]
```



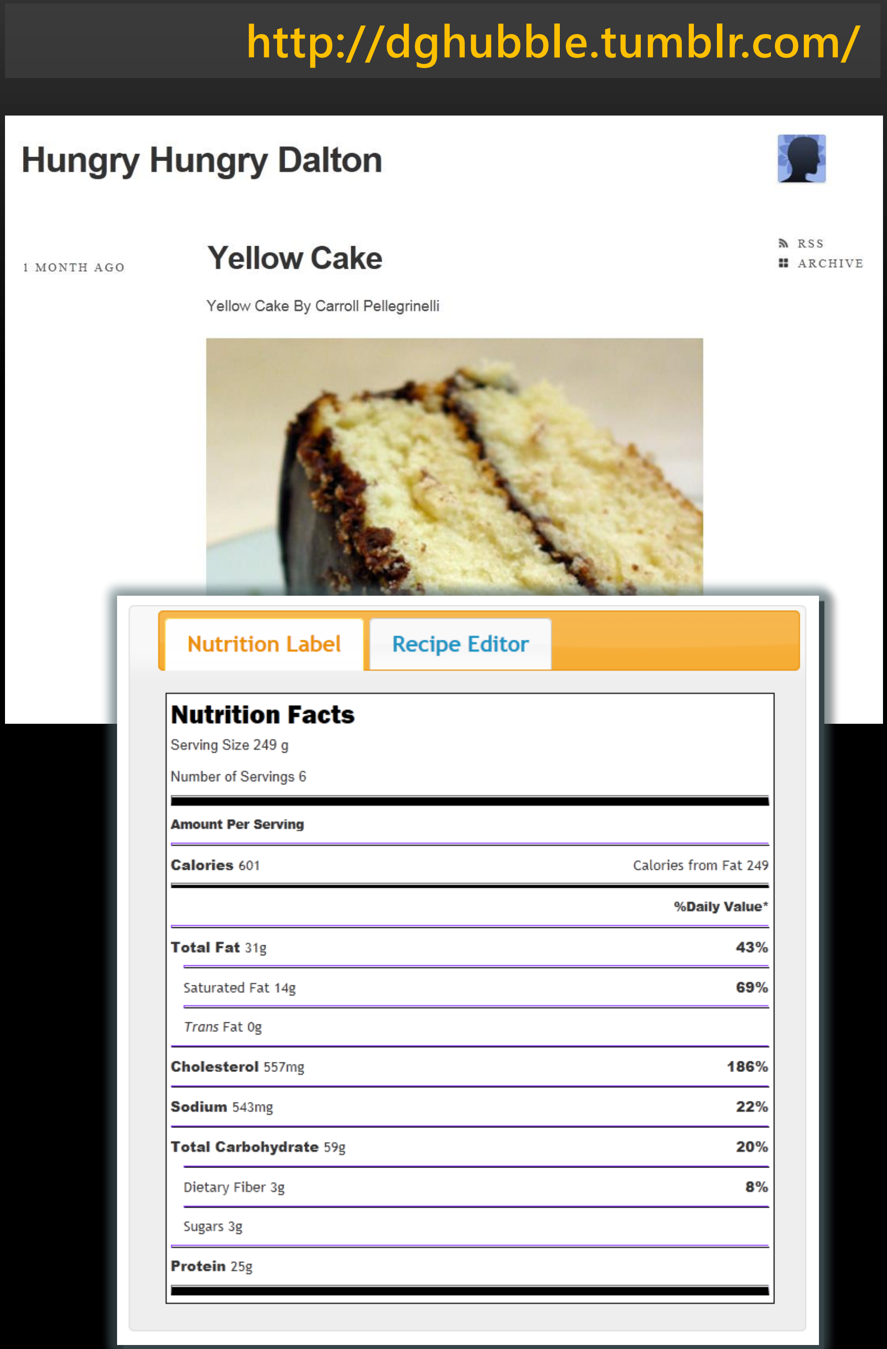
serving size	Times	368.02	Gram
total calories	Times	309.72	Calorie
fat calories	Times	58.413	Calorie
total fat	Times	6.546	Gram
% daily total fat	Times	9.8998	Percent
saturated fat	Times	1.5959	Gram
% daily saturated fat	Times	7.5358	Percent
trans fat	0		
cholesterol	Times	0.044422	Gram
% daily cholesterol	Times	14.795	Percent
sodium	Times	1.7696	Gram
% daily sodium	Times	73.6	Percent
total carbohydrates	Times	53.543	Gram
% daily carbohydrates	Times	17.71	Percent
dietary fiber	Times	6.8463	Gram
%daily dietary fiber	Times	25.73	Percent
sugars	Times	5.8525	Gram
protein	Times	12.1	Gram
% daily protein	Times	2.7658	Percent
vitamin A	Times	13.434	Percent
vitamin C	Times	107.62	Percent
calcium	Times	11.903	Percent
iron	Times	19.675	Percent

REFINEMENTS

CSS for formatting results as a label

UI for editing recipes

Remoting expressions



http://dghubble.tumblr.com/

Hungry Hungry Dalton

1 MONTH AGO

Yellow Cake

Yellow Cake By Carroll Pellegrinelli

Nutrition Facts

Serving Size 249 g

Number of Servings 6

Amount Per Serving	
Calories 601	Calories from Fat 249
	%Daily Value*
Total Fat 31g	**43%**
Saturated Fat 14g	**69%**
Trans Fat 0g	
Cholesterol 557mg	**186%**
Sodium 543mg	**22%**
Total Carbohydrate 59g	**20%**
Dietary Fiber 3g	**8%**
Sugars 3g	
Protein 25g	

UNITS OF MEASURE ARE EVERYWHERE

Finance

```
{ Euro -> 1.24 * Dollar,
  Yuan -> 0.15715 * Dollar, ... }
```

```
TradeTrigger[priceSpread_] := ...
```

Transportation

```
{ Mile   -> 1.609344 * Kilo * Meter,
  Gallon -> 3.785412 * Liter, ... }
```

```
Convert[22 * Mile / Gallon, ...
~~>     0.10692 Liter / Kilo*Meter
```

Textiles / Apparel

```
{ American -> 2 * British / 6,
  British  -> European / 6, ... }
```

Calendars

Engineering

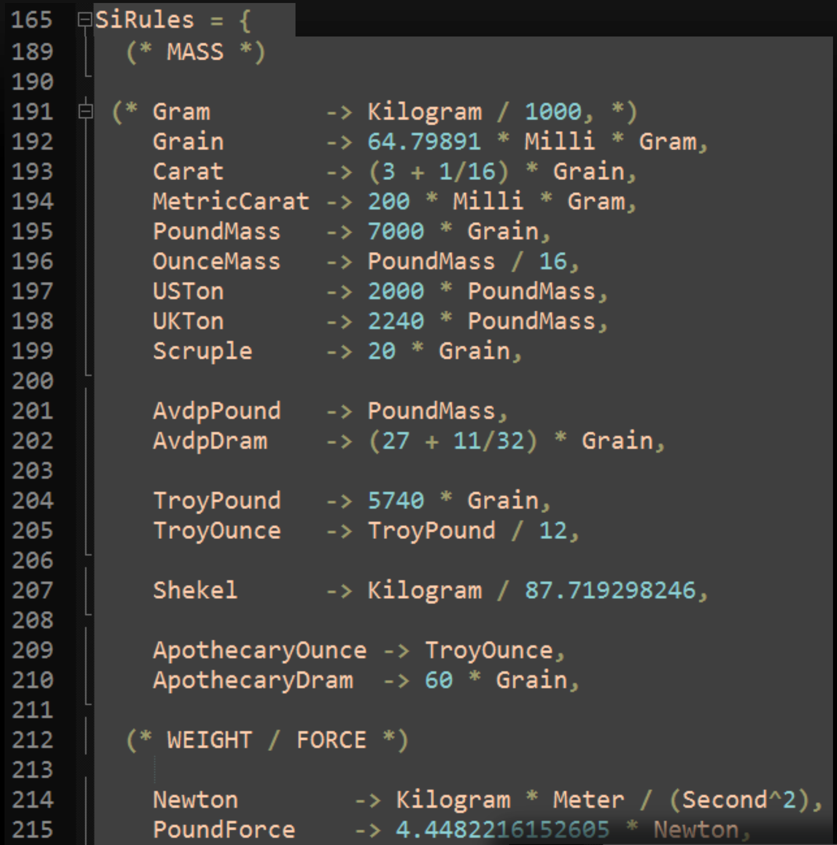
Geospatial

Agriculture

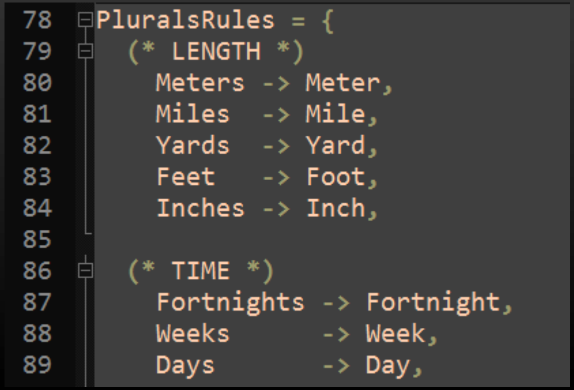
Government

Education

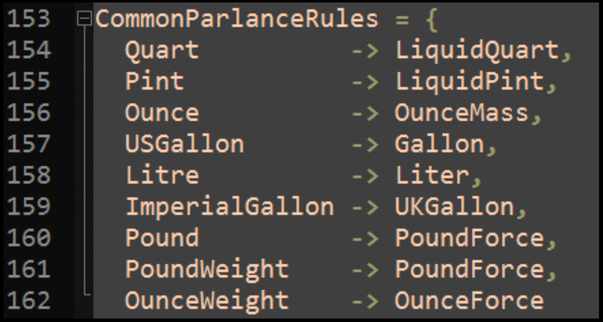
UNITS CONVERSION IN GENERAL



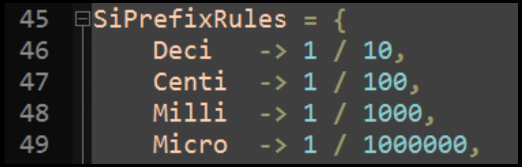
```
SiRules = {
  (* MASS *)

 (* Gram        -> Kilogram / 1000, *)
    Grain       -> 64.79891 * Milli * Gram,
    Carat       -> (3 + 1/16) * Grain,
    MetricCarat -> 200 * Milli * Gram,
    PoundMass   -> 7000 * Grain,
    OunceMass   -> PoundMass / 16,
    USTon       -> 2000 * PoundMass,
    UKTon       -> 2240 * PoundMass,
    Scruple     -> 20 * Grain,

    AvdpPound   -> PoundMass,
    AvdpDram    -> (27 + 11/32) * Grain,

    TroyPound   -> 5740 * Grain,
    TroyOunce   -> TroyPound / 12,

    Shekel      -> Kilogram / 87.719298246,

    ApothecaryOunce -> TroyOunce,
    ApothecaryDram  -> 60 * Grain,

  (* WEIGHT / FORCE *)

    Newton        -> Kilogram * Meter / (Second^2),
    PoundForce    -> 4.4482216152605 * Newton,
    KilogramForce -> 9.80665
    OunceForce    -> PoundFor
```



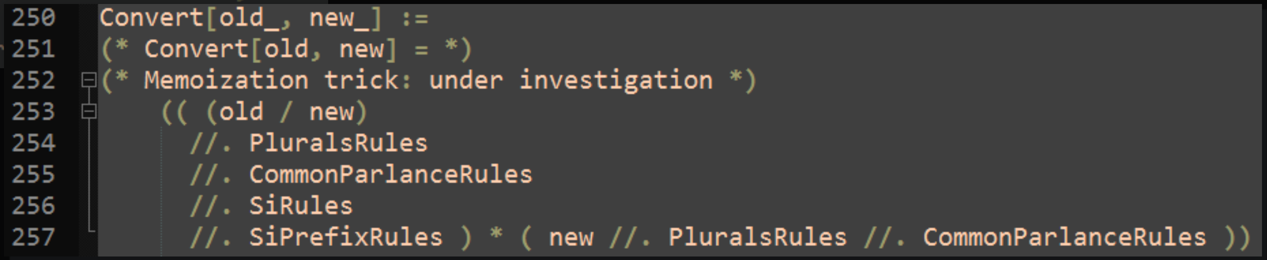
```
PluralsRules = {
  (* LENGTH *)
    Meters  -> Meter,
    Miles   -> Mile,
    Yards   -> Yard,
    Feet    -> Foot,
    Inches  -> Inch,

  (* TIME *)
    Fortnights -> Fortnight,
    Weeks      -> Week,
    Days       -> Day,
```



```
CommonParlanceRules = {
  Quart            -> LiquidQuart,
  Pint             -> LiquidPint,
  Ounce            -> OunceMass,
  USGallon         -> Gallon,
  Litre            -> Liter,
  ImperialGallon   -> UKGallon,
  Pound            -> PoundForce,
  PoundWeight      -> PoundForce,
  OunceWeight      -> OunceForce
```



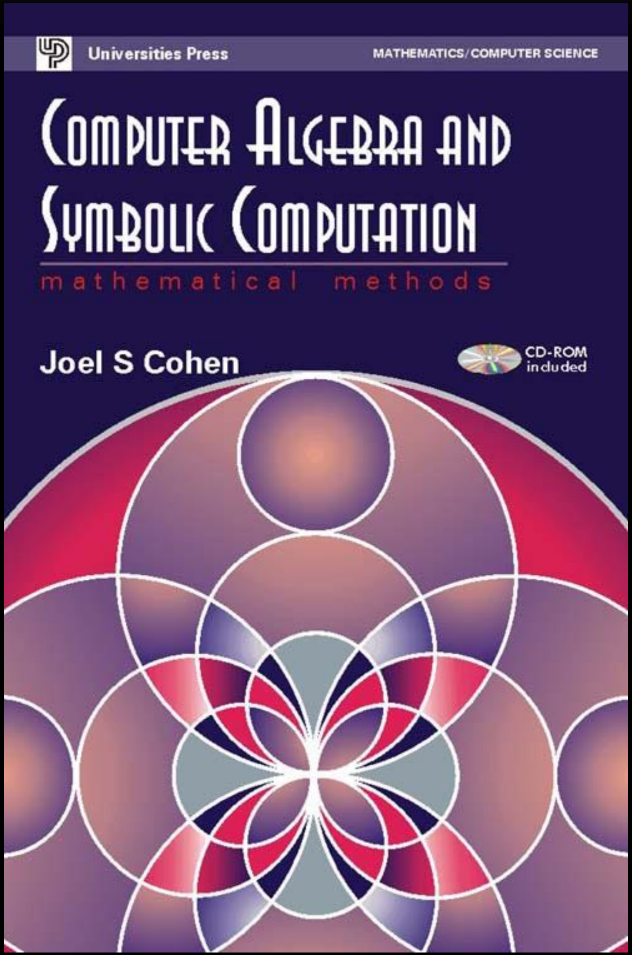
```
SiPrefixRules = {
    Deci   -> 1 / 10,
    Centi  -> 1 / 100,
    Milli  -> 1 / 1000,
    Micro  -> 1 / 1000000,
```



```
Convert[old_, new_] :=
(* Convert[old, new] = *)
(* Memoization trick: under investigation *)
    (( (old / new)
      //. PluralsRules
      //. CommonParlanceRules
      //. SiRules
      //. SiPrefixRules ) * ( new //. PluralsRules //. CommonParlanceRules ))
```

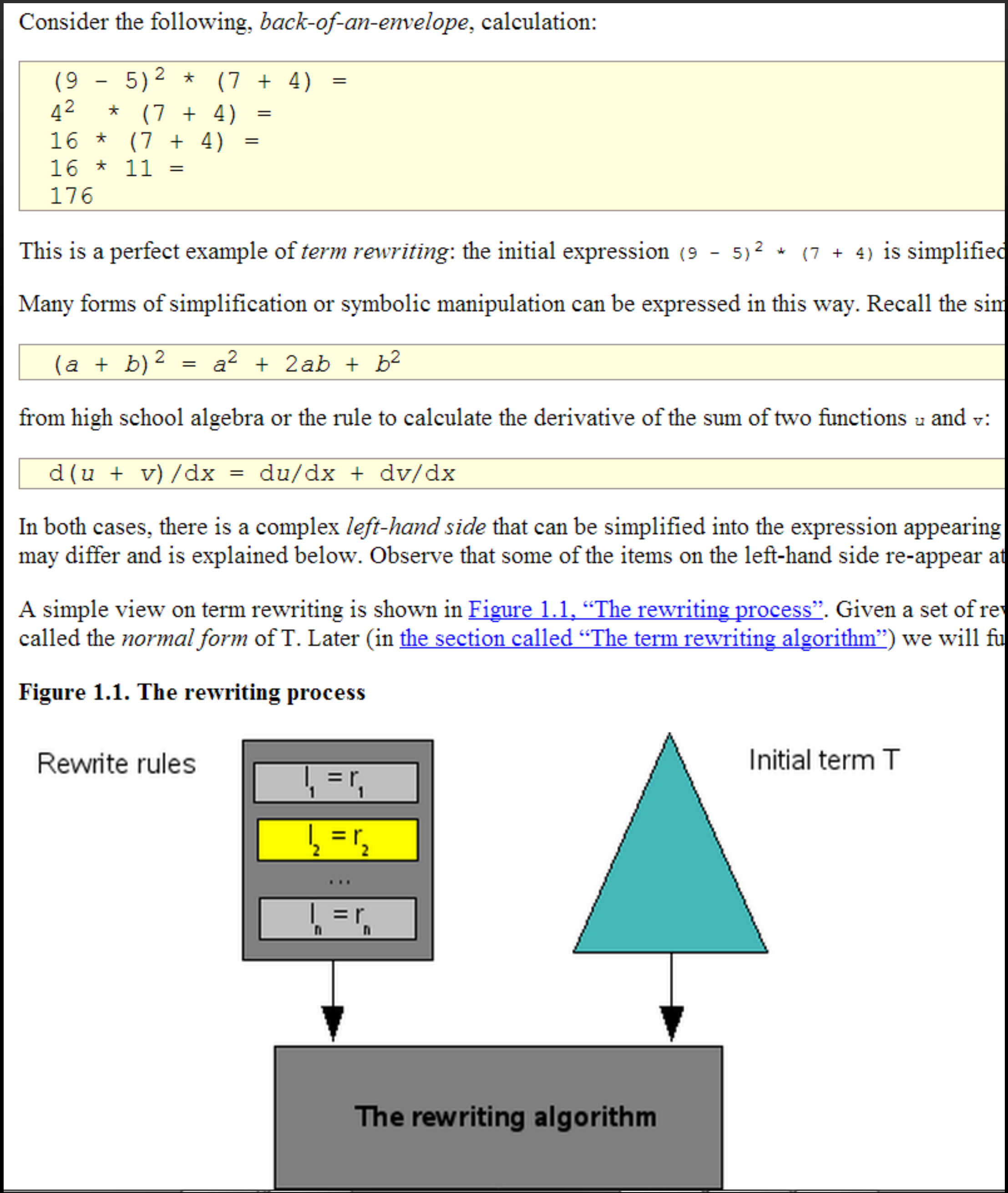
How to implement such an expression evaluator?

A bit of background reading, and…



Universities Press
MATHEMATICS/COMPUTER SCIENCE
Computer Algebra and Symbolic Computation
mathematical methods
Joel S Cohen
CD-ROM included

http://bit.ly/SGwWuh



Consider the following, *back-of-an-envelope*, calculation:

$$
\begin{aligned}
&(9 - 5)^2 * (7 + 4) = \\
&4^2 * (7 + 4) = \\
&16 * (7 + 4) = \\
&16 * 11 = \\
&176
\end{aligned}
$$

This is a perfect example of *term rewriting*: the initial expression $(9 - 5)^2 * (7 + 4)$ is simplified

Many forms of simplification or symbolic manipulation can be expressed in this way. Recall the sim

$$(a + b)^2 = a^2 + 2ab + b^2$$

from high school algebra or the rule to calculate the derivative of the sum of two functions u and v:

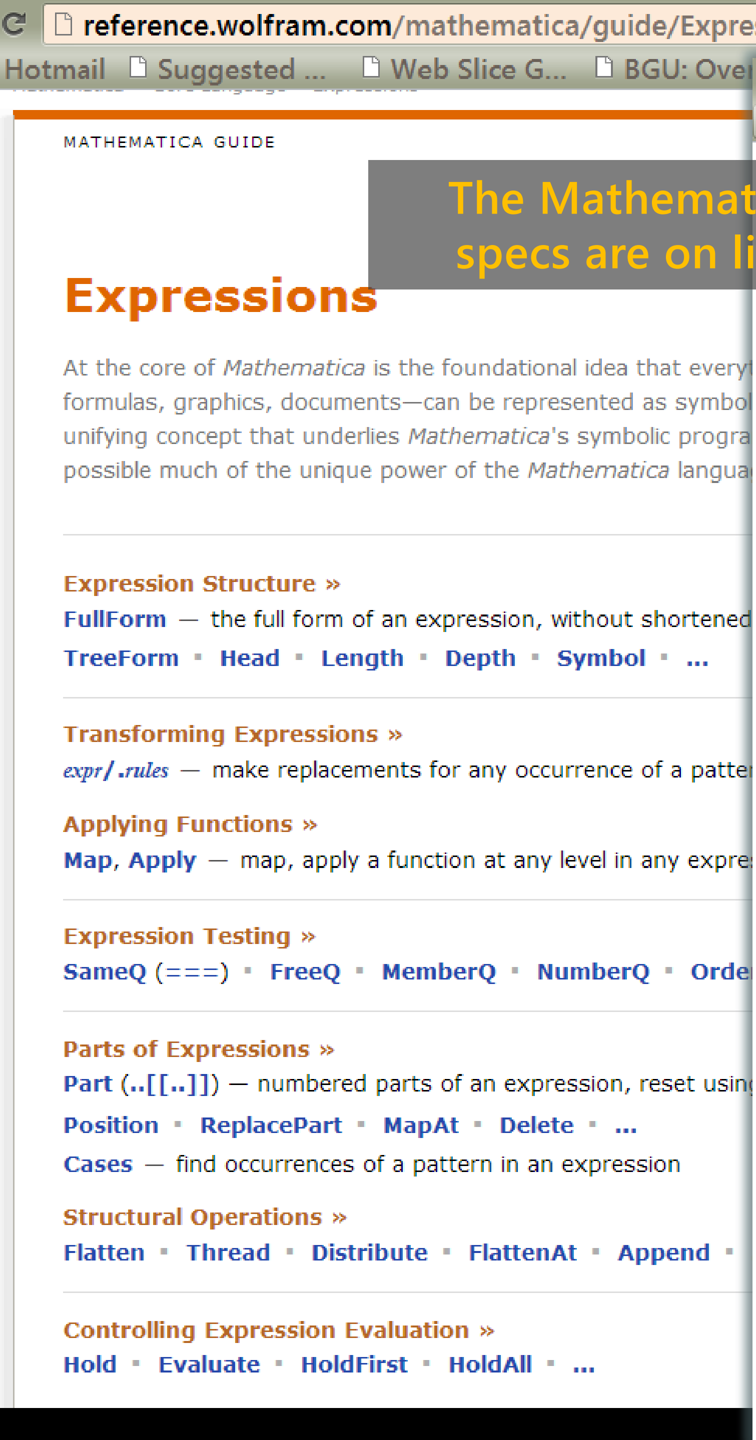
$$d(u + v)/dx = du/dx + dv/dx$$

In both cases, there is a complex *left-hand side* that can be simplified into the expression appearing may differ and is explained below. Observe that some of the items on the left-hand side re-appear at

A simple view on term rewriting is shown in Figure 1.1, “The rewriting process”. Given a set of rev called the *normal form* of T. Later (in the section called “The term rewriting algorithm”) we will fu

Figure 1.1. The rewriting process

The Mathematica specs are on line



MATHEMATICA GUIDE

Expressions

At the core of *Mathematica* is the foundational idea that every formulas, graphics, documents—can be represented as symbol unifying concept that underlies *Mathematica*'s symbolic progra possible much of the unique power of the *Mathematica* langua

Expression Structure »

FullForm — the full form of an expression, without shortened

TreeForm · **Head** · **Length** · **Depth** · **Symbol** · **...**

Transforming Expressions »

expr **/.** *rules* — make replacements for any occurrence of a patte

Applying Functions »

Map, **Apply** — map, apply a function at any level in any expre

Expression Testing »

SameQ (===) · **FreeQ** · **MemberQ** · **NumberQ** · **Orde**

Parts of Expressions »

Part (..[[..]]) — numbered parts of an expression, reset usin

Position · **ReplacePart** · **MapAt** · **Delete** · **...**

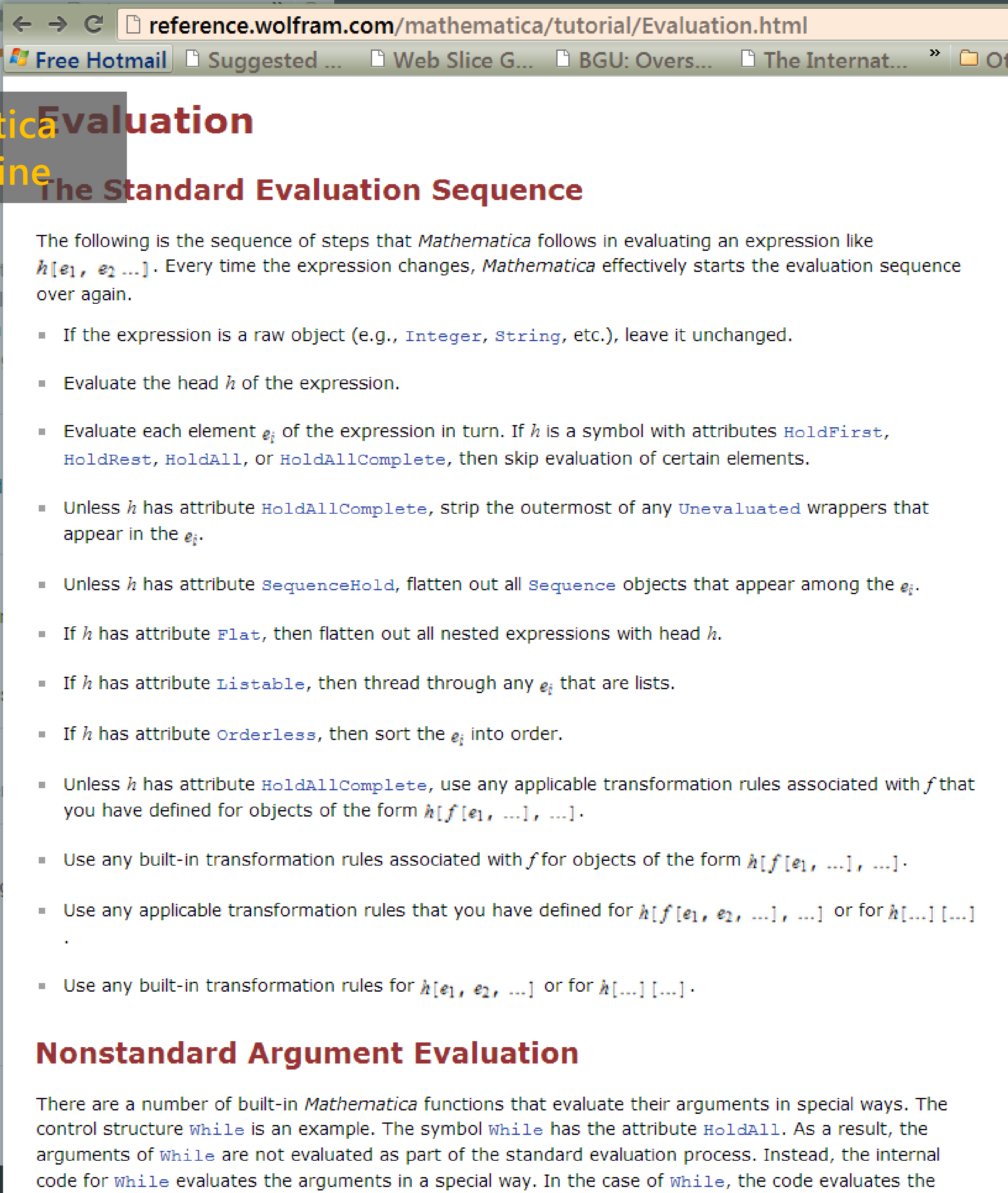
Cases — find occurrences of a pattern in an expression

Structural Operations »

Flatten · **Thread** · **Distribute** · **FlattenAt** · **Append** ·

Controlling Expression Evaluation »

Hold · **Evaluate** · **HoldFirst** · **HoldAll** · **...**



Evaluation

The Standard Evaluation Sequence

The following is the sequence of steps that *Mathematica* follows in evaluating an expression like $h[e_1, e_2 \ldots]$. Every time the expression changes, *Mathematica* effectively starts the evaluation sequence over again.

- If the expression is a raw object (e.g., `Integer`, `String`, etc.), leave it unchanged.
- Evaluate the head h of the expression.
- Evaluate each element e_i of the expression in turn. If h is a symbol with attributes `HoldFirst`, `HoldRest`, `HoldAll`, or `HoldAllComplete`, then skip evaluation of certain elements.
- Unless h has attribute `HoldAllComplete`, strip the outermost of any `Unevaluated` wrappers that appear in the e_i.
- Unless h has attribute `SequenceHold`, flatten out all `Sequence` objects that appear among the e_i.
- If h has attribute `Flat`, then flatten out all nested expressions with head h.
- If h has attribute `Listable`, then thread through any e_i that are lists.
- If h has attribute `Orderless`, then sort the e_i into order.
- Unless h has attribute `HoldAllComplete`, use any applicable transformation rules associated with f that you have defined for objects of the form $h[f[e_1, \ldots], \ldots]$.
- Use any built-in transformation rules associated with f for objects of the form $h[f[e_1, \ldots], \ldots]$.
- Use any applicable transformation rules that you have defined for $h[f[e_1, e_2, \ldots], \ldots]$ or for $h[\ldots][\ldots]$.
- Use any built-in transformation rules for $h[e_1, e_2, \ldots]$ or for $h[\ldots][\ldots]$.

Nonstandard Argument Evaluation

There are a number of built-in *Mathematica* functions that evaluate their arguments in special ways. The control structure `While` is an example. The symbol `While` has the attribute `HoldAll`. As a result, the arguments of `While` are not evaluated as part of the standard evaluation process. Instead, the internal code for `While` evaluates the arguments in a special way. In the case of `While`, the code evaluates the

REPRESENTATION FIRST

A JSON representation for every kind of expression

There are only two kinds:
Atoms and non-Atoms, a.k.a., Normal Forms
e.g., *headExpr* [*partExpr1*, *partExpr2*, …]

```
engine.CreateAtom(42).toJSON()
{number: {subtype: "Integer", value: 42}}
```

```
engine.CreateAtom(3.14159).toJSON()
{number: {subtype: "Real", value: 3.14159}}
```

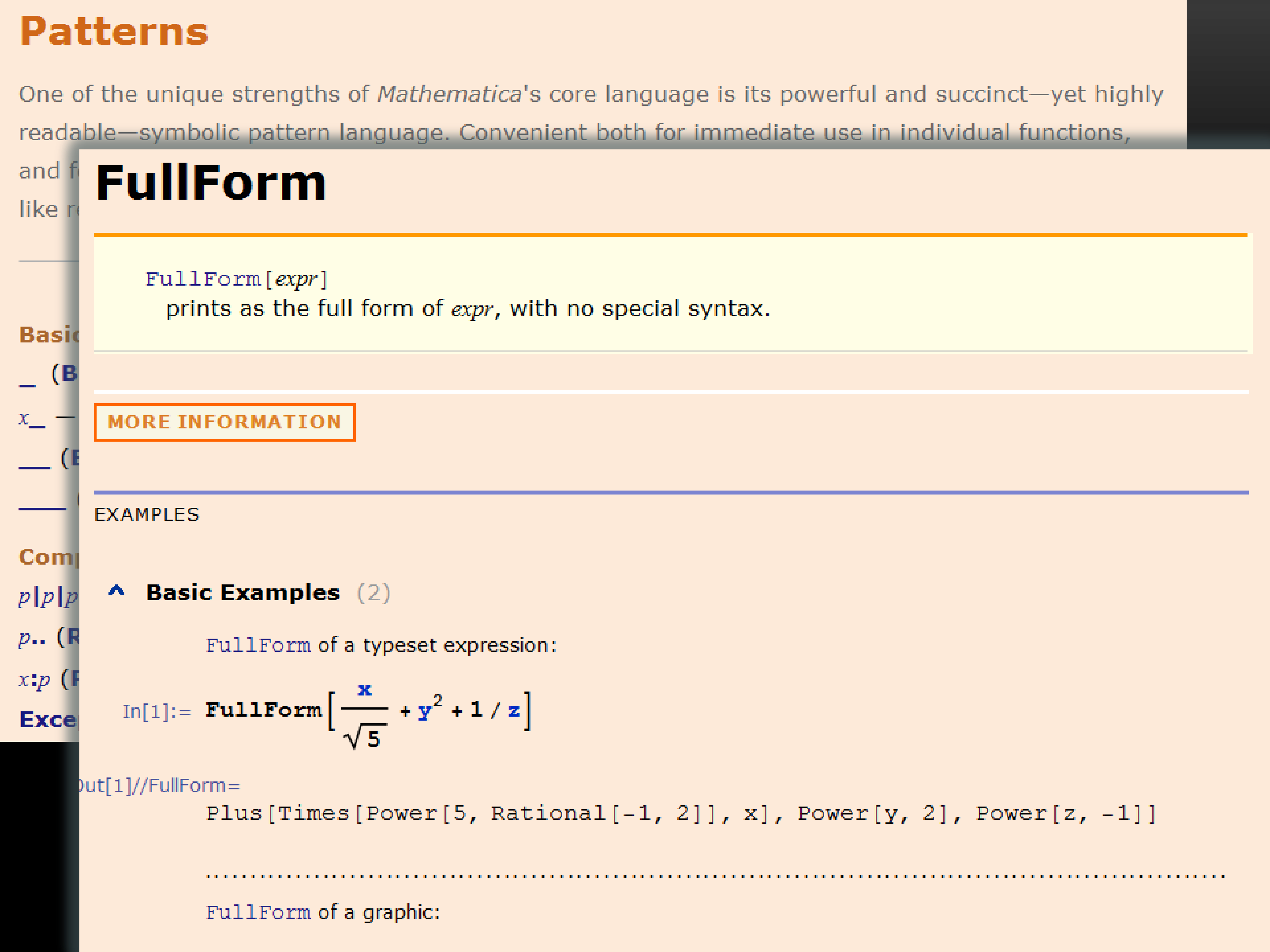
```
engine.CreateAtom('"myString"').toJSON()
{string: "\"myString\""}
```

```
engine.CreateAtom("mySymbol").toJSON()
{symbol: "mySymbol"}
```

```
engine.Evaluate("{\"a\", b, c -> 3}")
{head: "List",
 parts: [
   {string: "\"a\""},
   {symbol: "b"},
   {head: "Rule",
    parts: [
      {symbol: "c"},
      {number: {subtype: "Integer", value: 3}}
   ]}
]}
```

Three kinds of Atoms:
Numbers, Strings, & Symbols.
Numbers are Integers, Real,
Complex, Rational (N.Y.I.)

Normal Forms are just
recursive structures: explicit
abstract syntax trees (ASTs)



Patterns

One of the unique strengths of *Mathematica*'s core language is its powerful and succinct—yet highly readable—symbolic pattern language. Convenient both for immediate use in individual functions,

and f

like r

Basic

_ (B

*x*_ —

__ (B

___ (

Comp

p|*p*|*p*

p.. (R

x:*p* (P

Exce

FullForm

`FullForm[`*expr*`]`
prints as the full form of *expr*, with no special syntax.

MORE INFORMATION

EXAMPLES

^ Basic Examples (2)

`FullForm` of a typeset expression:

In[1]:= **FullForm**$\left[\frac{x}{\sqrt{5}} + y^2 + 1 / z\right]$

Out[1]//FullForm=

```
Plus[Times[Power[5, Rational[-1, 2]], x], Power[y, 2], Power[z, -1]]
```

`FullForm` of a graphic:

ENGINEERING THE EVAL ENGINE

Bootstrap gradually

- "Never more than a few minutes away from something that works"
- Focus on scenario-prioritized functions
- Implementation flows from the representation shown

Don't be distracted by syntax

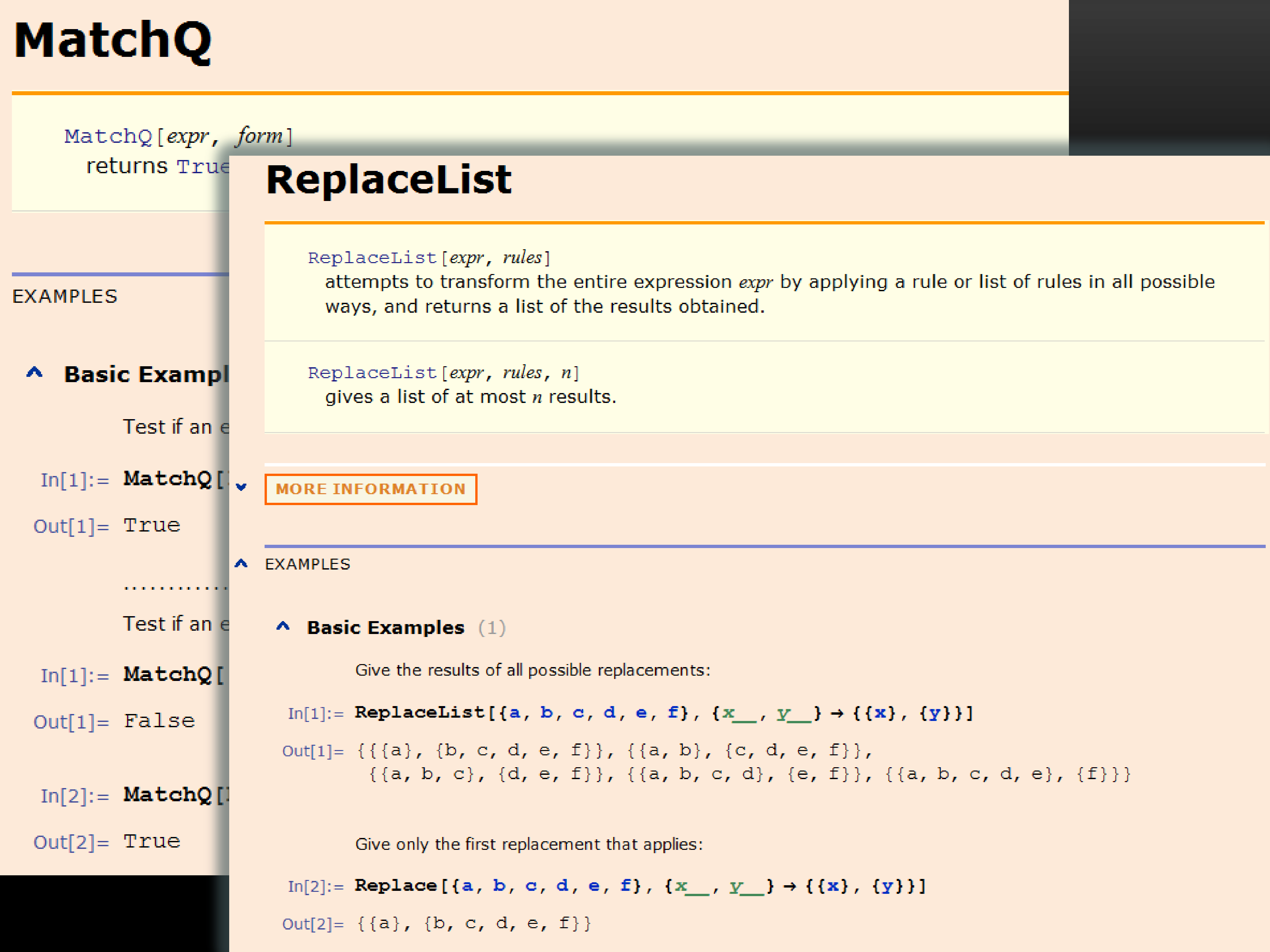
- Bootstrap with FullForm ASTs & JSON reps
- Use Jison to implement parser incrementally

Start with patterns and replacement

- Patterns, MatchQ, ReplaceList, ReplaceAll, Set, SetDelayed

Remoting is almost free

- Run exactly the same engine in Node.JS and in the browser
- Most of the challenge is on the Mathematica side
  - Quoting expressions is sometimes non-trivial



MatchQ

MatchQ[*expr*, *form*]
returns True

EXAMPLES

Basic Exampl

Test if an e

```
In[1]:= MatchQ[
Out[1]= True
```

....

Test if an e

```
In[1]:= MatchQ[
Out[1]= False

In[2]:= MatchQ[
Out[2]= True
```

ReplaceList

ReplaceList[*expr*, *rules*]
attempts to transform the entire expression *expr* by applying a rule or list of rules in all possible ways, and returns a list of the results obtained.

ReplaceList[*expr*, *rules*, *n*]
gives a list of at most *n* results.

MORE INFORMATION

EXAMPLES

Basic Examples (1)

Give the results of all possible replacements:

```
In[1]:= ReplaceList[{a, b, c, d, e, f}, {x__, y__} → {{x}, {y}}]

Out[1]= {{{a}, {b, c, d, e, f}}, {{a, b}, {c, d, e, f}},
         {{a, b, c}, {d, e, f}}, {{a, b, c, d}, {e, f}}, {{a, b, c, d, e}, {f}}}
```

Give only the first replacement that applies:

```
In[2]:= Replace[{a, b, c, d, e, f}, {x__, y__} → {{x}, {y}}]

Out[2]= {{a}, {b, c, d, e, f}}
```

REMOTING

Move Expressions to the Data

- Flexible expressions means "elastic platform" – scenarios need not be pre-canned
- Evaluator is a sandbox – not necessary to host all of JavaScript
- AFFINITY and PRIVACY are the drivers for remoting
- e.g., Evaluate the NFL vector computation on a server with the ingredients
- e.g., Evaluate business offers on the client device without moving private data

CONCLUSIONS

Term-Rewriting is an underutilized but generally useful and powerful computing strategy

- Robust Units-of-Measure is a huge risk-reducer
- Concise patterns-and-rules specs is a cost-reducer
- Remotable Expressions are a new kind of data resource

It is surprisingly easy to implement in standard web-computing settings

Established tools & practice (e.g., Mathematica) substantially increase attractiveness of the approach

ESSENTIAL CONTRIBUTORS

Avi Bar-Zeev, Elad Ben-Israel, Steve Coast, Elad Gerson,
Yair Gheva, Gur Kimchi, Limor Lahiani, Erik Meijer,
Kanchan Mitra, Savas Parastiditis

SPONSORS

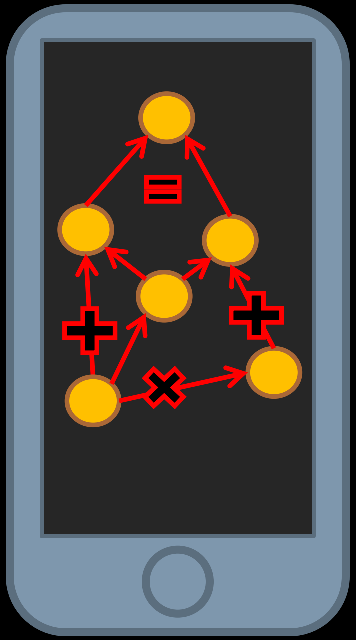
IPE Core Platform

Bimal Mehta, James Whittaker, Gurdeep Singh Pall, Qi Lu

ACKNOWLEDGEMENTS

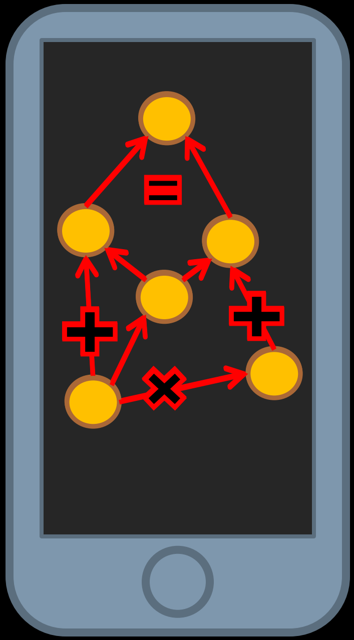
OVERFLOW SLIDES

REMOTING PROBLEM



Moving data to the computation is not the best use of $data plan$

REMOTING SOLUTION



Move the computation to the data

JaqSON

Remoting format for Jacquard expressions

MORE ONLINE SCENARIOS

- *Get me to the airport on time*
  - Reactively monitor traffic, flight info, current location
- *Average age of singers? Maximum salary of CEOs?*
  - Map-reduce style; statistics on-the-fly
- *Help me by a car*
  - Distributed workflow with privacy concerns