“Trick or Treat” Protocols

- Trick-or-Treater must convince victim that she poses a credible threat
- Need to \textit{prove} you know are a qualified tricker
- But, revealing trickiness shouldn’t allow victim to impersonate a tricker

\textit{Exception around Halloween, this is called an “authentication protocol”}.

Cryptographic Hash Functions

\textbf{One-way}

Given $h$, it is hard to find $x$ such that $H(x) = h$.

\textbf{Collision resistance}

Given $x$, it is hard to find $y \neq x$ such that $H(y) = H(x)$. 
Example One-Way-ish Function

**Input:** two 100 digit numbers, \( x_1 \) and \( x_2 \)

**Output:** the middle 100 digits of \( x_1 \times x_2 \)

Given \( x = (x_1, x_2) \): easy to calculate \( f(x) \).

"Easy" means there is a procedure with running time in \( O(N^2) \) where \( N \) is number of digits

Given \( h = f(x) \): hard to find an \( z \) such \( h = f(z) \).

"Hard" means (we hope) the fastest possible procedure has running time in \( \Omega(2^N) \).

---

Trick-or-Treat

"Trick or Treat?", \( H \) (secret)

Trickers?

\( R = H \) (secret + \( Z \))

Valid!

Trickers Bureau

You know what you need to survive tomorrow...we'll talk more about how this works on the web later.

---

Trick-or-Treating without Calling the Tricker’s Bureau

\( R = Z \) + \( H \) (secret)

Valid!

Trickers Bureau

### Building a Language

**Design the grammar**

What strings are in the language?

Use BNF to describe all the strings in the language

**Make up the evaluation rules**

Describe what every string in the language means

**Build an evaluator**

Implement a procedure that takes a string in the language as input and an environment and outputs its value:

\[ \text{meval: String} \times \text{Environment} \rightarrow \text{Value} \]

---

Implementing Interpreters

Problem Set 7: posted today, due Monday 9 November.

- Understand the Charme interpreter (described in Chapter 11)
- Modify it to create a new language
Is this an exaggeration?

It is no exaggeration to regard this as the most fundamental idea in programming:

The evaluator, which determines the meaning of expressions in the programming language, is just another program.

To appreciate this point is to change our images of ourselves as programmers. We come to see ourselves as designers of languages, rather than only users of languages designed by others.

Abelson and Sussman, Structure and Interpretation of Computer Programs (p. 360)

Building an Evaluator

To build an evaluator we need to:

– Figure out how to represent data in programs
  What is a procedure, frame, environment, etc.
  – Implement the evaluation rules
    For each evaluation rule, define a procedure that follows the behavior of that rule.

Next: we’ll look at a high-level how the application rule is implemented
Next week and Chapter 11: detailed walk-through of the interpreter

Core of the evaluator: meval

def meval(expr, env):
    if isPrimitive(expr):
        return evalPrimitive(expr)
    elif isIf(expr):
        return evalIf(expr, env)
    elif isDefinition(expr):
        evalDefinition(expr, env)
    elif isName(expr):
        return evalName(expr, env)
    elif isLambda(expr):
        return evalLambda(expr, env)
    elif isApplication(expr):
        return evalApplication(expr, env)
    else:
        error ('Unknown expression type: ' + str(expr))

Stateful Application Rule

To apply a constructed procedure:

1. Construct a new environment, whose parent is the environment of the applied procedure.
2. For each procedure parameter, create a place in the frame of the new environment with the name of the parameter. Evaluate each operand expression in the environment of the application and initialize the value in each place to the value of the corresponding operand expression.
3. Evaluate the body of the procedure in the newly created environment. The resulting value is the value of the application.

Eval and Apply are defined in terms of each other.

Eval and Apply are defined in terms of each other.

def evalApplication(expr, env):
    subexprs = expr
    subexprvals = map(lambda sexpr: meval(sexpr, env), subexprs)
    return mapply(subexprvals[0], subexprvals[1])
To apply a constructed procedure:
1. Construct a new environment, whose
   parent is the environment of the
   applied procedure.
2. For each procedure parameter, create
   a place in the frame of the new
   environment with the name of the
   parameter. Evaluate each operand
   expression in the environment of the
   application and initialize the value in
   each place to the value of the
   corresponding operand expression.
3. Evaluate the body of the procedure in
   the newly created environment. The
   resulting value is the value of the
   application.

```python
def mapply(proc, operands):
    if isPrimitiveProcedure(proc): ...
    elif isinstance(proc, Procedure):
        params = proc.getParams()
        newenv = Environment(proc.getEnvironment())
        ...
```

Charge

- Read Chapter 11
- PS7 posted today, due Monday, Nov 9

Remember to make “Trick-or-Treaters” to solve your challenge!