Overview

Applying Formal Methods: Part II

Overview

Apply formal methods to implement the Set data type and solve the model-checking problem through the evaluation function implemented.

- Implement and test the Set data type.
- Represent a Kripke model in Haskell programs and implement its evaluation function.
- Use the evaluation function to solve the model checking problem.

Overview

The End

The Set Data Type III

Properties of Sets

Properties of Sets

- A ∩ B = B ∩ A (commutative law)
- (A ∩ B) ∩ C = A ∩ (B ∩ C) (associative law)
- φ ∩ A = $\underbrace{\varphi,}$ U ∩ A = A (law of $\underbrace{\varphi}$ and $\underbrace{U}_{\text{universe}}$)   φ empty set
- A ∩ A = A (idempotent law)
- A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) (distributive law)
  - That is, ∩ distributes over ∪.

property checking
/testing

Practice

Implement a test in the Haskell language to demonstrate that set intersection is commutative.

A B ~~C~~

$$(A \cap B) = (B \cap A)$$

Cases	A	B	Input	Output pair
⋮				True false.

Condition $A \cap B = B \cap A$.

Set data type

primarily using Recursion

primarily use list comprehension

The Set Data Type III: Properties of Sets

The End

The Set Data Type III

Preparing Tests

Tests

- Input-output pairs (black box test)
- A helper function (white box test)
- Single operation (e.g., union, intersection etc.)
- Sequence of operations
- Known properties (e.g., associativity of intersections)
- Properties that are not always true

Sequence of operations

$$\begin{tikzcd}
\text{Empty Set} \arrow[r, "\overset{f}{op\ 1}"] & \text{Set 1} \arrow[r, "\overset{g}{op\ 2}"] \arrow[d] & \text{Set 2} \arrow[r, "\overset{h}{op 3}"] \arrow[d] & \text{Set 3} \\
\text{Card} = 0 & \text{Card} = 2 & \text{Card} = 4 & \text{Card} = 5
\end{tikzcd}$$

$h \circ g \circ f$

Testing Properties

Construct single case or use randomly generated instances

- Single operation

$$A \cup A = A, \quad A \cap A = A$$

- Sequence of operations (known to be true)

$$for\ any\ set\ A, \quad \text{card}\,(A \cup A) = \text{card}\,(A \cap A)$$

$$for\ any\ set\ A, B, \qquad (A \cup B) \cap C = (A \cap C) \cup (B \cap C)$$

Testing Properties (cont.)

- Properties that are not always true

$$for\ any\ set\ A, B, \qquad \mathrm{card}\ (A \cup B) = \mathrm{card}\ (A)$$

$$for\ any\ set\ A, B, and\ C, \quad (A - B) - C = A - (B - C)$$ ← (*)

- What will our random testing tool respond to in these test?

False when $A = \{1\}$ $B = \phi$ $C = \{1\}$ $B - C = \phi$

L.H.S. $A - B = \{1\}$ $A - (B - c)$

— $(A - B) - C = \{\ \}$ $= \{1\}$

→ Finding counterexamples

thru. quickCheck.

helps to debug the ~~problem~~ program.

The Set Data Type III: Preparing Tests

The End

The Set Data Type IV

Implement Set Data Type via Other Paradigm

Other Programming Paradigms

- Procedural languages
- Object oriented languages
- Any other paradigms other than functional programming paradigm

Project

By using a concrete example, describe and discuss how to associate, convert, and apply the software development skills learned in the functional programming framework to other programming frameworks.

The Set Data Type

- Commonly used in other languages (e.g., Java, C, etc.)
- From a Haskell implementation example (e.g., the module Set we developed), it includes the following aspects:
    - Design, specification, implementation, testing, verification
    - What will you do if you are asked to associate and convert your Haskell program (design or code) to an implementation using a language with another programming paradigm?

Some Discussion Items

- Program correctness
- Formulate precise specification
- Rapid prototyping
- Operations includes higher-order functions
- Tools available for automatic property testing

The Set Data Type IV

The End

The Evaluation Function I

Representation of Kripke Models

Kripke Structures

A *Kripke structure* $\mathcal{M}$ is a three-tuple $\langle W, I, J \rangle$, where:

- Set — W is a nonempty set, whose elements are called *worlds*.
- function — $I : \textbf{PropVar} \to \boxed{\mathcal{P}(W)}$ is an *interpretation* function that maps each propositional variable p to a set of worlds.
- function — $J : \textbf{PName} \to \boxed{\mathcal{P}(W \times W)}$ is a function that maps each principal name A into a relation on worlds (i.e., a subset of $W \times W$).

Intuition: "Truth tables" for modal logic!

$$(\underset{\underset{W}{\uparrow}}{\quad}, \underset{\underset{I}{\uparrow}}{\quad}, \underset{\underset{J}{\uparrow}}{\quad})$$

- $\underline{I}$ is the Kripke-equivalent of a truth assignment: $I(p)$ is set of worlds in which p is true.
- $\underline{J(A)}$ is relation describing how A views various worlds: each pair $(w, w') \in J(A)$ indicates that, when the current world is w, principal A believes current world might be w'.

$$\underline{E}_M :: (W, I, J) \to \text{Form} \to \underline{\text{true}}$$

"Initial thought"

Kripke Semantics

Each Kripke structure $\mathcal{M} = \langle W, I, J \rangle$ gives rise to a function

$$\mathcal{E}(M) = \quad \mathcal{E}_{\mathcal{M}}[\![-]\!] : \textbf{Form} \to \mathcal{P}(W),$$

$\mathcal{P}(W)$

$\mathcal{E} :: \mathcal{M} \to \text{Form} \to ..$

W is set

representation.

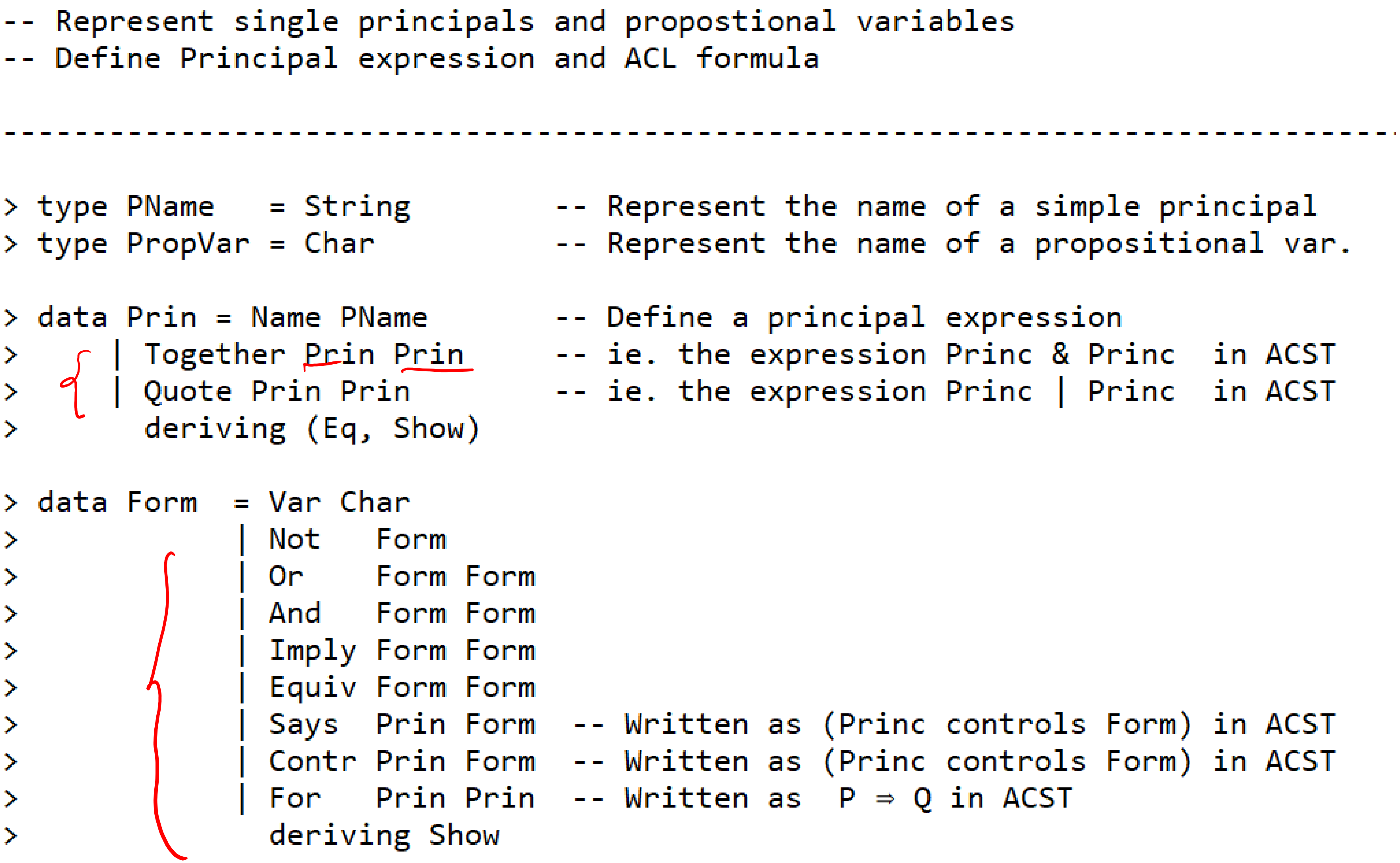
where $\mathcal{E}_{\mathcal{M}}[\![\varphi]\!]$ is the set of worlds in which φ is true.

- $\mathcal{E}_{\mathcal{M}}[\![-]\!]$ is the Kripke-equivalent of rules for truth tables.
- $\mathcal{E}_{\mathcal{M}}[\![\varphi]\!]$ defined based on the structure of φ, plus the individual components of $\mathcal{M} = \langle W, I, J \rangle$.
- $\mathcal{M}$ satisfies φ provided that $\mathcal{E}_{\mathcal{M}}[\![\varphi]\!] = W$.
- φ is a tautology provided that every Kripke structure satisfies φ.

$\mathcal{E}(M, \varphi) = W$

(Model Check problem
$M \models \varphi$?)

Access-Control Formulas



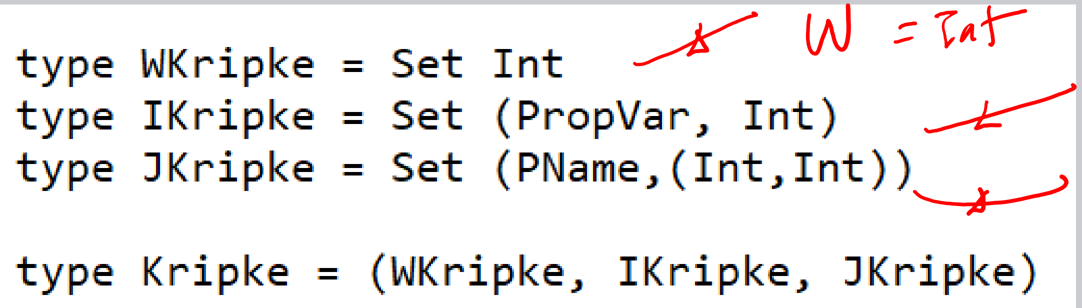
```
-- Represent single principals and propostional variables
-- Define Principal expression and ACL formula

------------------------------------------------------------------------------

> type PName   = String        -- Represent the name of a simple principal
> type PropVar = Char          -- Represent the name of a propositional var.

> data Prin = Name PName       -- Define a principal expression
>     | Together Prin Prin     -- ie. the expression Princ & Princ  in ACST
>     | Quote Prin Prin        -- ie. the expression Princ | Princ  in ACST
>       deriving (Eq, Show)

> data Form  = Var Char
>            | Not   Form
>            | Or    Form Form
>            | And   Form Form
>            | Imply Form Form
>            | Equiv Form Form
>            | Says  Prin Form  -- Written as (Princ controls Form) in ACST
>            | Contr Prin Form  -- Written as (Princ controls Form) in ACST
>            | For   Prin Prin  -- Written as  P ⇒ Q in ACST
>              deriving Show
```

Kripke Models

Mathematical definition

- $\mathcal{M} = (W, I, J)$
- W: nonempty set
- $I : \text{PropVar} \rightarrow \mathcal{P}(W)$
- $J : \text{PName} \rightarrow \mathcal{P}(W \times W)$

Haskell definition



```
type WKripke = Set Int
type IKripke = Set (PropVar, Int)
type JKripke = Set (PName,(Int,Int))

type Kripke = (WKripke, IKripke, JKripke)
```

W = Int

W I J

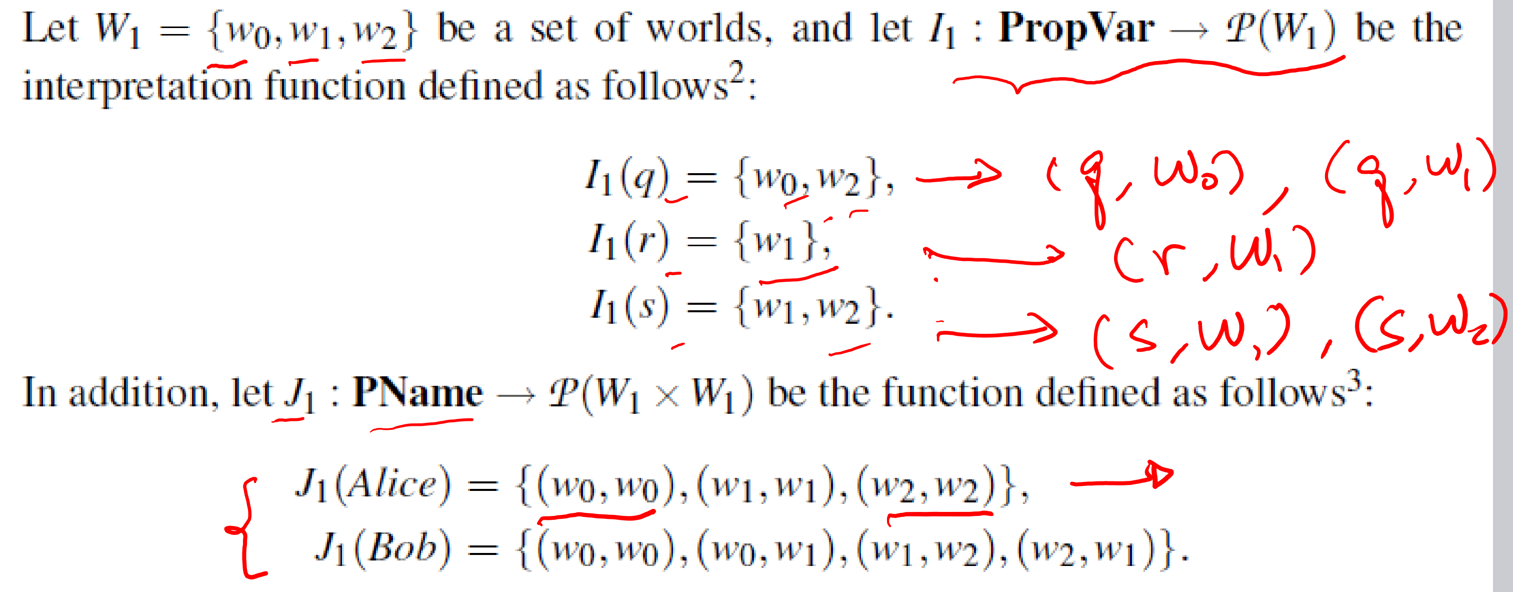
Kripke Models (cont.)

Remarks

- The functions *I* and *J* in the definition are not directly translated from the mathematical definition. Both are now a list of tuples that can be coded more easily.

Example 2.8

Mathematical definition



Let $W_1 = \{w_0, w_1, w_2\}$ be a set of worlds, and let $I_1 : \mathbf{PropVar} \to \mathcal{P}(W_1)$ be the interpretation function defined as follows[2]:

$$I_1(q) = \{w_0, w_2\}, \longrightarrow (q, w_0), (q, w_1)$$

$$I_1(r) = \{w_1\}, \longrightarrow (r, w_1)$$

$$I_1(s) = \{w_1, w_2\}. \longrightarrow (s, w_1), (s, w_2)$$

In addition, let $J_1 : \mathbf{PName} \to \mathcal{P}(W_1 \times W_1)$ be the function defined as follows[3]:

$$\left\{\begin{aligned} J_1(Alice) &= \{(w_0, w_0), (w_1, w_1), (w_2, w_2)\}, \longrightarrow \\ J_1(Bob) &= \{(w_0, w_0), (w_0, w_1), (w_1, w_2), (w_2, w_1)\}. \end{aligned}\right.$$

↑

World View

Haskell definition

I_1 = The set where the underlying list is [(q,w0), (q, w2), (r, w1), (s, w1), (s, w2)]

J_1 = The set where the underlying list is [(Alice, (w0, w0)),... , (Alice, (w2, w2)), (Bob, (w0, w0)),... , (Bob, (w2, w1))]

The Evaluation Function I: Representation of Kripke Models

The End

The Evaluation Function I

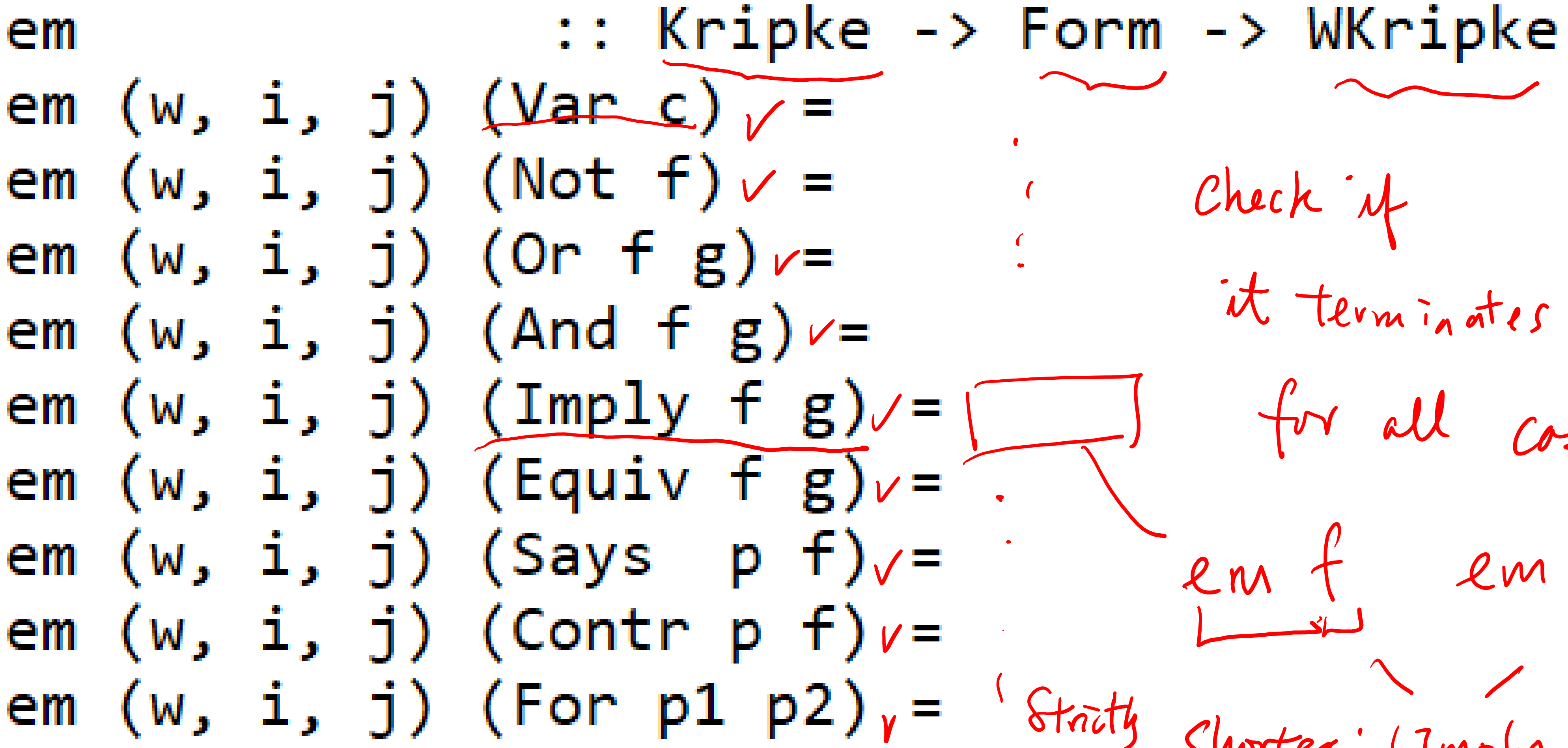
Evaluation Function: Specification

Semantics: Definition of $\mathcal{E}_{\mathcal{M}}[\![-]\!]$

$$
\begin{aligned}
\mathcal{E}_{\mathcal{M}}[\![p]\!] &= I(p) \\
\mathcal{E}_{\mathcal{M}}[\![\neg\varphi]\!] &= W - \mathcal{E}_{\mathcal{M}}[\![\varphi]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \wedge \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \vee \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] &= (W - \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!]) \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \equiv \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2 \supset \varphi_1]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_{\mathcal{M}}[\![P \text{ says } \varphi]\!] &= \{w \mid J(P)(w) \subseteq \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]\} \\
\mathcal{E}_{\mathcal{M}}[\![P \text{ controls } \varphi]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \text{ says } \varphi) \supset \varphi]\!] \\
\mathcal{E}_{\mathcal{M}}[\![P \Rightarrow Q]\!] &= \begin{cases} W, & \text{if } J(Q) \subseteq J(P) \\ \emptyset, & \text{otherwise} \end{cases}
\end{aligned}
$$

Evaluation Function: Specification



```
                          Model          A.C.L formula
em                     :: Kripke      -> Form        -> WKripke
em (w, i, j) (Var c) ✓ =
em (w, i, j) (Not f) ✓ =
em (w, i, j) (Or f g) ✓ =
em (w, i, j) (And f g) ✓ =
em (w, i, j) (Imply f g) ✓ =
em (w, i, j) (Equiv f g) ✓ =
em (w, i, j) (Says  p f) ✓ =
em (w, i, j) (Contr p f) ✓ =
em (w, i, j) (For p1 p2) ✓ =
```

Check if it terminates for all cases.

em f   em g

Strictly Shorter (Imply f g)

Evaluation Function: Remarks

- The mathematical definition specified on the left is translated to fill the blanks on the right side of the code.
- In the actual implementation, several helper functions are added to simplify the code.

The Evaluation Function I: Evaluation Function (Specification)

The End

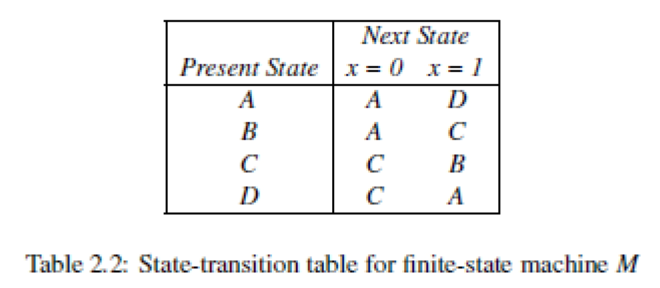
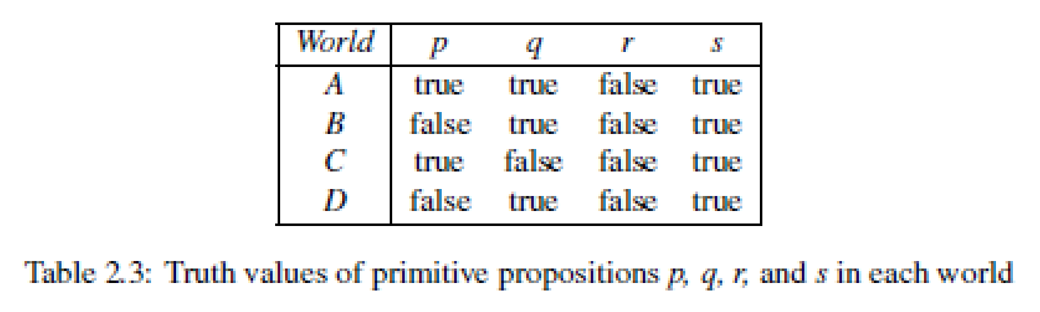
The Evaluation Function II

Preparing Tests

Tests

- The implementation involves several helper functions, and we can prepare tests for these functions.
- In the coming demonstration, we will focus on the following text examples: 2.8, 2.12 and 2.15.
- Also review the example 2.9 (not included in the demonstration).

Example 2.9

Model $\mathcal{M} = (W, I, J)$	**Supporting information**
$W = \{A, B, C, D\}$ $I(p) = \{A, C\}$, $I(q) = \{A, B, D\}, I(r)$ $= \varnothing, I(s)$ $= \{A, B, C, D\}$	 

	Next State	
Present State	*x = 0*	*x = 1*
A	*A*	*D*
B	*A*	*C*
C	*C*	*B*
D	*C*	*A*

Table 2.2: State-transition table for finite-state machine *M*

World	*p*	*q*	*r*	*s*
A	true	true	false	true
B	false	true	false	true
C	true	false	false	true
D	false	true	false	true

Table 2.3: Truth values of primitive propositions *p*, *q*, *r*, and *s* in each world

Given an observer o, the relation J(o) is specified by the state transition table in table 2.2.

The Evaluation Function II

The End

Solving the Model-Checking Problem

Remarks

$\mathcal{M} \vDash \varphi\,?\ versus\ \vDash \varphi?$

$\mathcal{M} \vDash \phi$?

- $\mathcal{M}$ = $(W, I\ , J)$
- $\mathcal{M} \vDash \phi$ ($\mathcal{M}\ satisfies\ \phi$); $i.e., \phi\ is\ true\ in\ any\ w\ in\ W$
- Remarks: $\vDash \varphi\ means\ \varphi\ is\ true\ for\ any\ model\ \mathcal{M}$

Semantics

The evaluation function $\mathcal{E}_{\mathcal{M}}$

- Defined by recursion
- Access-control operators (e.g., says, controls, and $\Rightarrow$) are defined, and their meanings are formulated according to the intended meaning of the access-control properties

Kripke Structures

- They are often used to analyze a variety of situations and provide semantics for modal and temporal logics, providing a basis for automated model checking.
- The language for access-control formulas can be enriched to describe security policies (ACST, Chapters 5, 13), and Kripke structure can be extended accordingly to define the semantics for the enriched formulas.

Solving the Model Checking Problem

The End

Weekly Summary

Applying Formal Methods II

Summary

Recap

- Develop the Set data type.
- Implement the evaluation function (solve the model checking problem).

Set Data Type

- Determine/select types and classes
- Essential operations and additional utilities
- Programming paradigms (recursion, list comprehensions)
- Formulate and execute tests: black box, white box and automatic property testing (QuickCheck)
- (*) Explore how to apply the methods used to an implementation of the data type in other programming framework

Evaluation Functions for Kripke Structures

The implementation involves:

- Choosing a presentation of the Kripke structure
- Following the specification and using the Set data type
- Dividing the programming task by using helper functions
- Testing the components of the overall evaluation functions
- Applying the function to solve the model-checking problem

Weekly Summary

The End