Lecture 4 - May 15

Overview of Compilation, Lexical Analysis

Example Compiler: Obj-Rel Bridge
Alphabets and Strings

Program Optimization Problem: Summary

external user's point of view

```
b := ... ; c := ... ; a := ...
across 1 |..| n is i
 loop
   read d
   a := a * 2 * b * c * d
 end
```

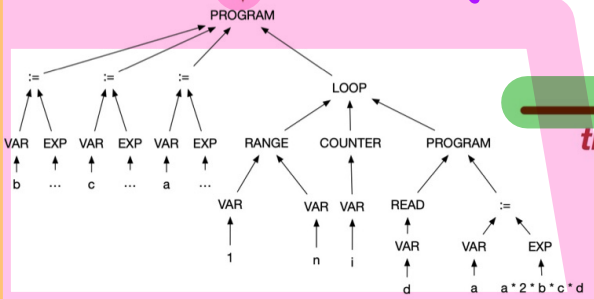
optimized

```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across 1 |..| n is i
 loop
   read d
   a := a * temp * d
 end
```

parsed

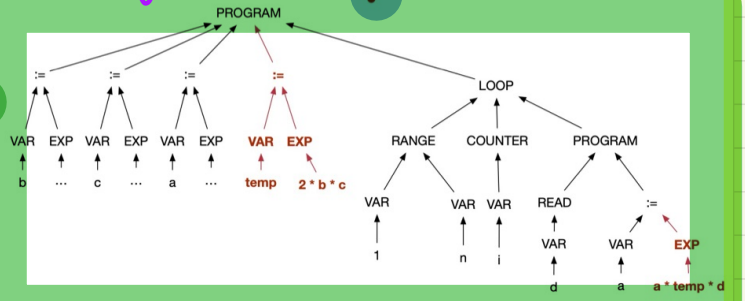
pretty-printed

prog1



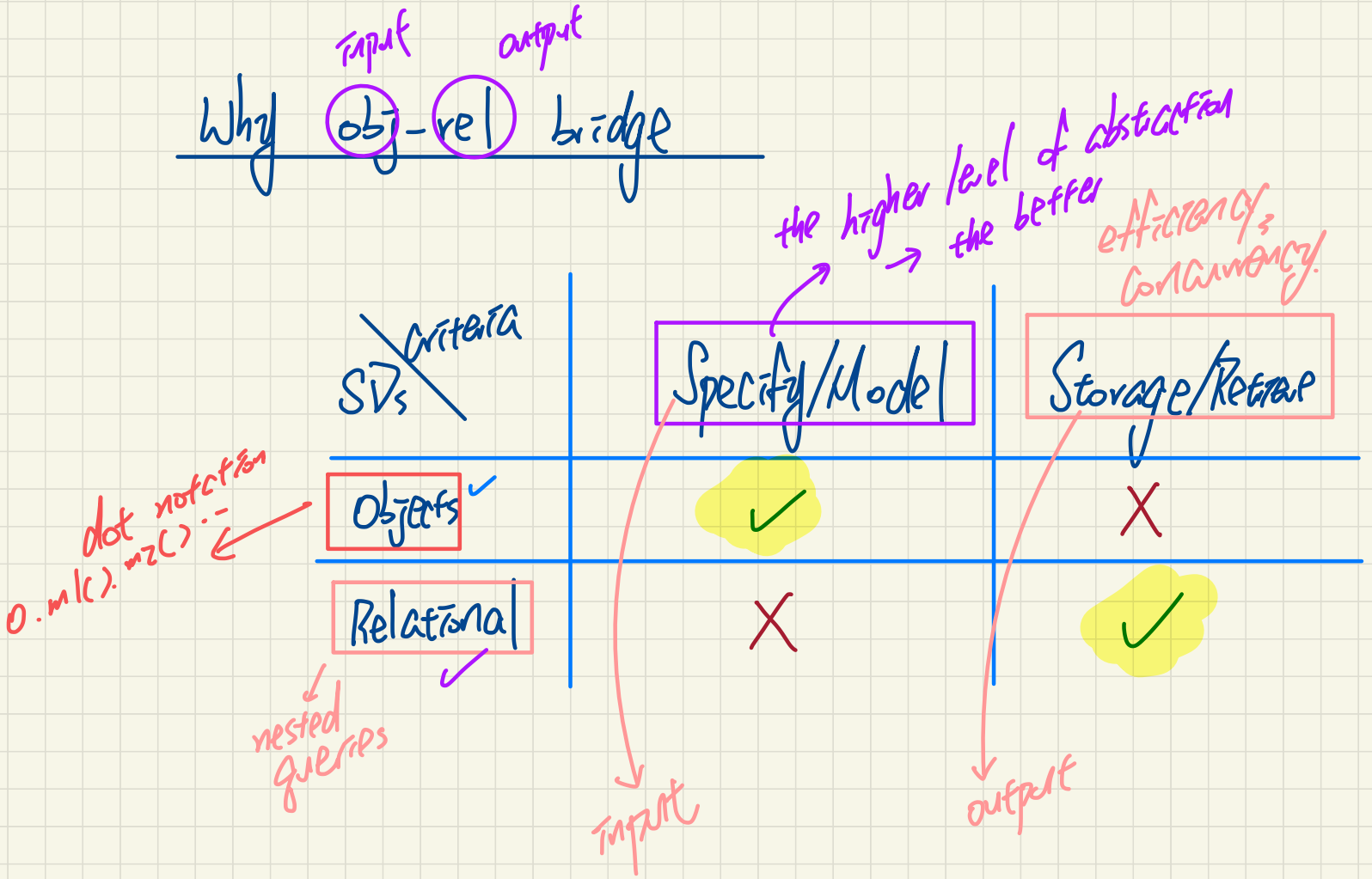
transformed

prog2

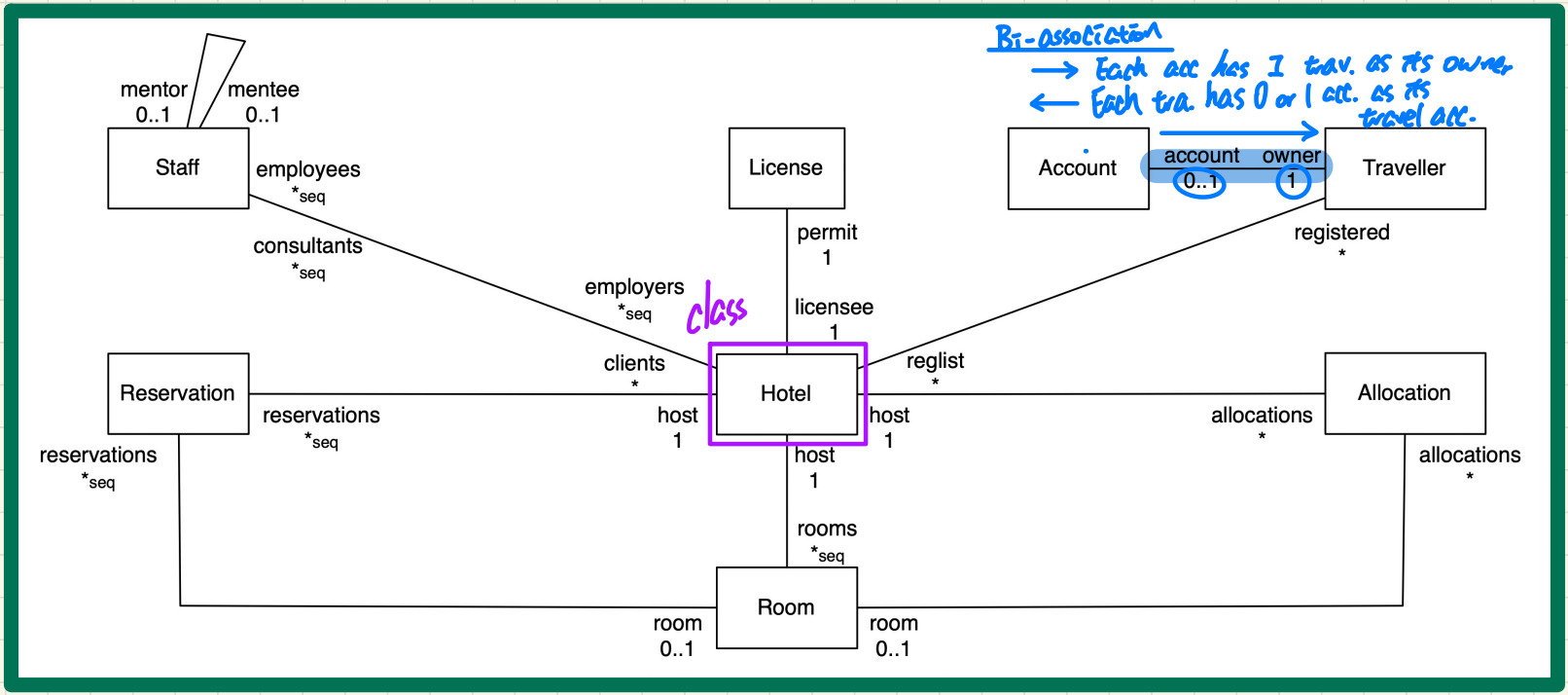


semantics-preserving transformations

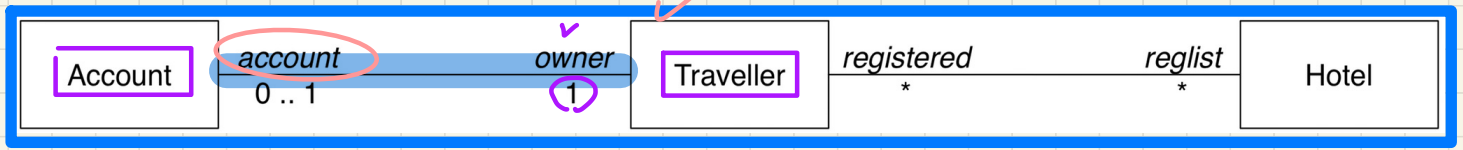
1. prog1 behaves same as prog2
2. prog2 more efficient than prog1


input
output
Why obj-rel bridge
the higher level of abstraction → the better
efficiency, concurrency
criteria
SDs
Specify/Model
Storage/Retrieve
Objects
Relational
dot notation
o.m1().m2()...
nested queries
input
output

Example Compiler 2: Data Model



Example Compiler 2: Source Program




```
class Account {
  attributes
    owner: Traveller . account
    balance: int
}
```

bi-assoc.

```
class Traveller {
  attributes
    name: string
    reglist: set(Hotel . registered)[*]
}
```

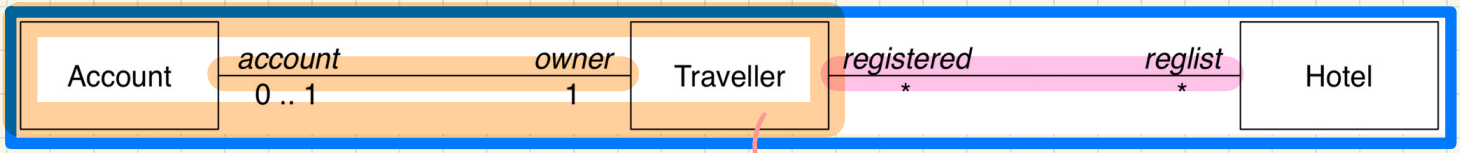
```
class Hotel {
  attributes
    name: string
    registered: set(Traveller . reglist)[*]
  methods
    register {
        t? : extent(Traveller)
      & t? /: registered
      ==>
          registered := registered \/ {t?}
      || t?.reglist := t?.reglist \/ {this}
    }
}
```

syntax of a DSL (def. using some CFG)

traveller

Example Compiler 2: Target Program

* Each bi-association is transformed into a table.



Account	
oid	balance
1	100

Traveller	
oid	name
2	alan
3	mark

Hotel	
oid	name
4	GLAD

Account_owner_Traveller_account		
oid	owner	account
5	3	1

Hotel_registered_Traveller_reglist		
oid	registered	reglist
6	2	4
7	3	4

object ids

Table Schemas

```
CREATE TABLE `Account`(
  `oid` INTEGER AUTO_INCREMENT, `balance` INTEGER,
  PRIMARY KEY (`oid`));
CREATE TABLE `Traveller`(
  `oid` INTEGER AUTO_INCREMENT, `name` CHAR(30),
  PRIMARY KEY (`oid`));
CREATE TABLE `Hotel`(
  `oid` INTEGER AUTO_INCREMENT, `name` CHAR(30),
  PRIMARY KEY (`oid`));
CREATE TABLE `Account_owner_Traveller_account`(
  `oid` INTEGER AUTO_INCREMENT, `owner` INTEGER, `account` INTEGER,
  PRIMARY KEY (`oid`));
CREATE TABLE `Traveller_reglist_Hotel_registered`(
  `oid` INTEGER AUTO_INCREMENT, `reglist` INTEGER, `registered` INTEGER,
  PRIMARY KEY (`oid`));
```

```
CREATE PROCEDURE `Hotel_register`(IN `this?` INTEGER, IN `t?` INTEGER)
  BEGIN
    ...
  END
```

Stored Procedures

Major challenge

Turn dot notation in Object-Base SL to select-queries in relational SL

Example Compiler 2: Path Transformation

Object Path

```
this.owner.reglist
```

in the context of Account class

this = Account

this.owner.reglist

Traveller

set of hotels

semantically equal.

Table Queries

```
SELECT (VAR 'reglist')
       (TABLE 'Hotel_registered_Traveller_reglist')
       (VAR 'registered' = (SELECT (VAR 'owner')
                           (TABLE 'Account_owner_Traveller_account')
                           (VAR 'owner' = VAR 'this')))
```

this.owner.

reglist

Account	
oid	balance
1	100

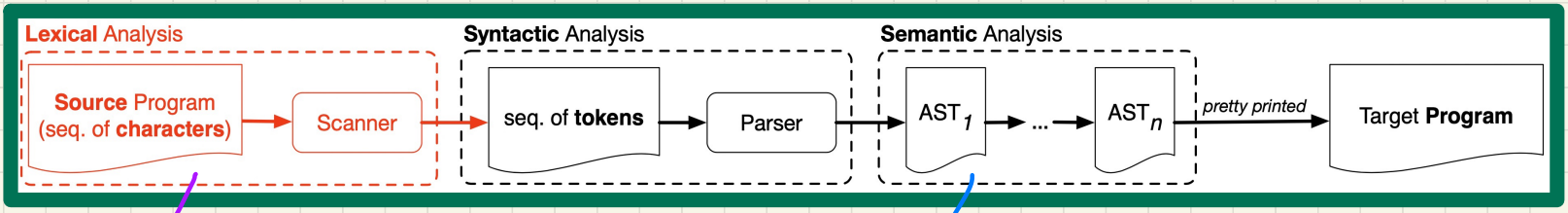
Traveller	
oid	name
2	alan
3	mark

Hotel	
oid	name
4	GLAD

Account_owner_Traveller_account		
oid	owner	account
5	3	1

Hotel_registered_Traveller_reglist		
oid	registered	reglist
6	2	4
7	3	4

Scanner in Context



Scan through input char by char

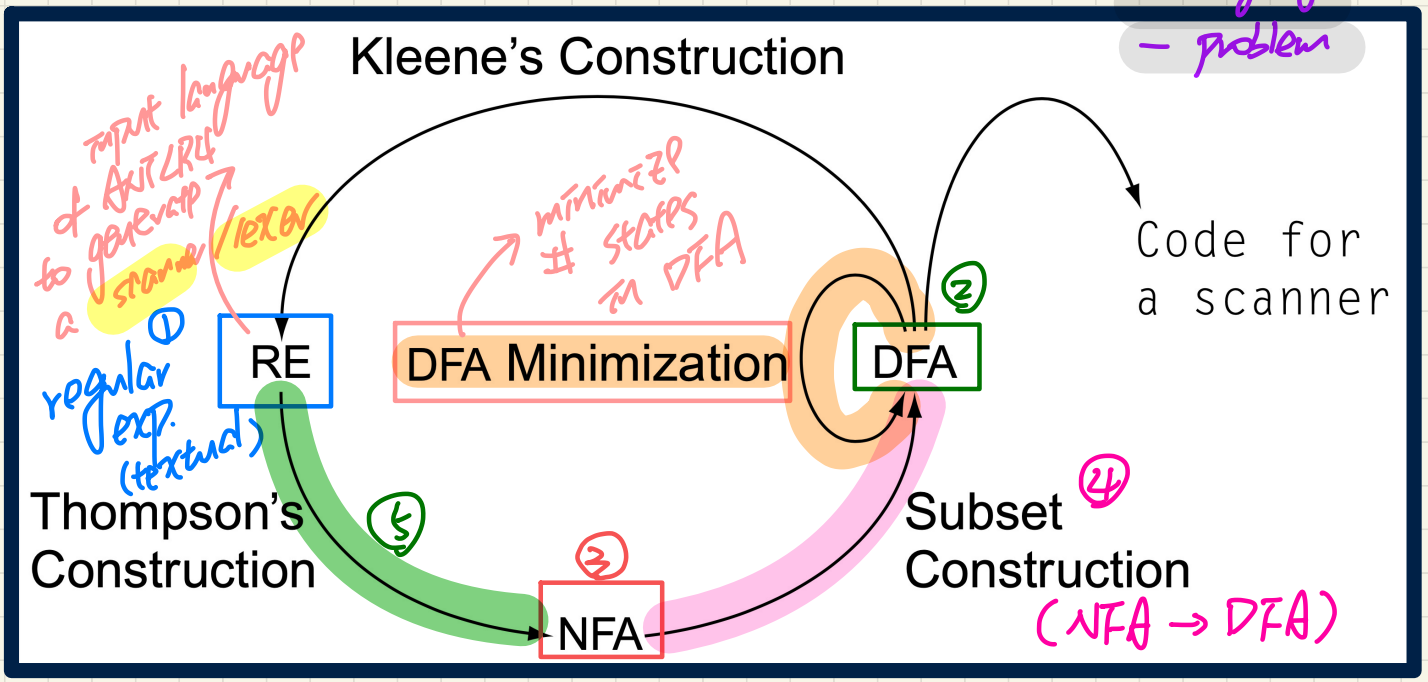
→ if all tokens recognized as valid patterns ↝ pass (seq. of token output)

→ if at least one token not recognized as a valid pattern ↝ fail (report lexical error)

[composite visitor

Scanner: Formulation & Implementation

- alphabet
- string
- language
- problem



Alphabet

non-empty set of characters

$\Sigma_{bin} = \{0, 1\}$ $|\Sigma_{bin}| = 2$

$\Sigma_{eng} = \{a, b, \ldots, z, A, B, \ldots, Z\}$ $|\Sigma_{eng}| = 52$

set enumerations

Set Comprehension

$\{e \mid P\}$ boolean condition / predicate involving e.

$\Sigma_{dec} = \{d \mid 0 \leq d \wedge d \leq 9\}$

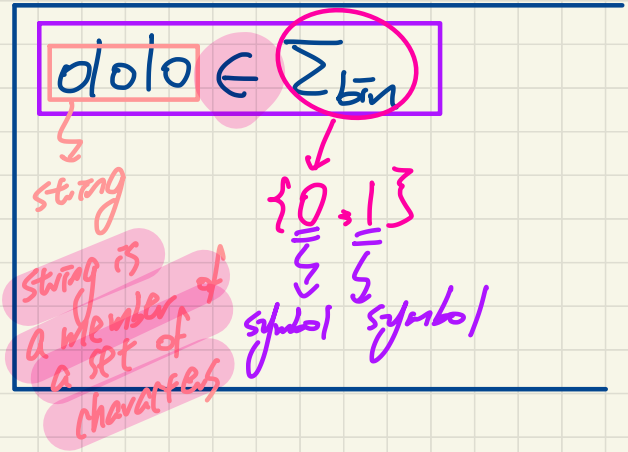
set comprehension

set of all such d that is in-between 0 and 9.

the resulting set contains all such e when the predicate P holds.

$\Sigma_{key} = \{A, B, \ldots, Z,$
$0, 1, 2, 3, \ldots 9,$
$!, @, \#, \$ \ldots\}$

keyboard alphabet



Empty String

ε s.t. $|\varepsilon| = 0$

$01010 \in \Sigma_{bin}$ is a string over Σ_{bin}

$01010 \in \Sigma_{bin}$

Identities

operation	identity	why?
string concat.	ε	$x\varepsilon = x = \varepsilon x$
addition	0	$a + 0 = a = 0 + a$
multiplication	1	$a * 1 = a = 1 * a$
conjunction $\wedge$	true	$p \wedge \text{true} \equiv p \equiv \text{true} \wedge p$
disjunction $\vee$	false	$p \vee \text{false} \equiv p \equiv \text{false} \vee p$