

(a) False. 正则语言可以是有限的也可以是无限的。例如，语言A = {a^n | n ≥ 0}，这是由所有a的幂组成的语言，它是无限的，但也是正则的。

(b) False. 一个语言如果有非确定有限自动机（NFA），并不意味着它就是非正则的。实际上，所有正则语言都有一个NFA。

(c) True. 每个非上下文无关的语言也是非正则的，因为上下文无关语言是正则语言的超集。

(d) False. 语言{a^n b^n | n ≥ 0}不是正则的，因此它没有一个正则表达式。a\*b\* 表示的是任意多个a后跟任意多个b，但这不包括严格成对出现的a和b。

(e) False. 如果B是上下文无关语言A是B的子集，这并不意味着A也是上下文无关的。A可能是正则的，甚至是平凡的，并不一定要是上下文无关的。

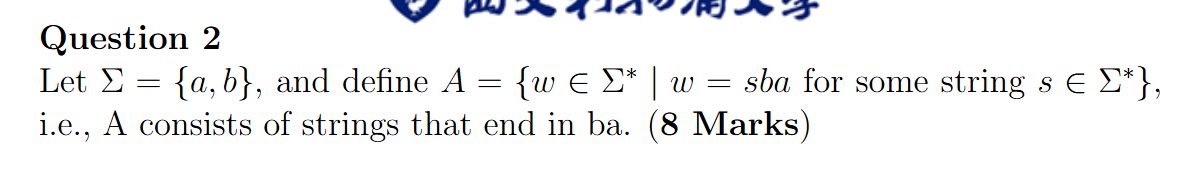
(f) False. 不是所有由非确定有限机（NPDA）识别的语言都能由确定有限自动机（DPDA）识别。存在一些非确定性的语言是DPDA无法识别的。

(g) True. 图灵可判定语言在连接下是封闭的，这意味着如果两个语言都是图灵可判定的，那么它们的连接也是图灵可判定的。

(h) True. 如果语言A可以被非确定图灵机（NTM）识别，那么A是图灵可识别。所有的非确定图灵机可识别的语言也是确定图灵机可识别的。

(i) False. 不是所有的上下文无关语言都是可判定的。实际上，上下文无关语言的可判定性是一个开放问题，有些上下文无关语言是递归不可判定的。

(j) True. 如果语言A可以通过映射归约到图灵可识别的语言B，并且B是可判定的，那么A也是可判定的。这是因为映射归约意味着A的问题可以转换成B的问题，而B的问题可以解决，因此A的问题也可以解决。



# Question 2的解释：

详细答案：

要描述这个语言，我们可以用正则表达式。正则表达式是一种用来描述字符串集合的数学表示方法，它可以定义一个语言中的所有字符串。对于语言 A ，我们想要匹配所有以 "ba" 结尾的字符串，前面的部分可以是任意数量的 'a' 和 'b' 字符。

正则表达式为：

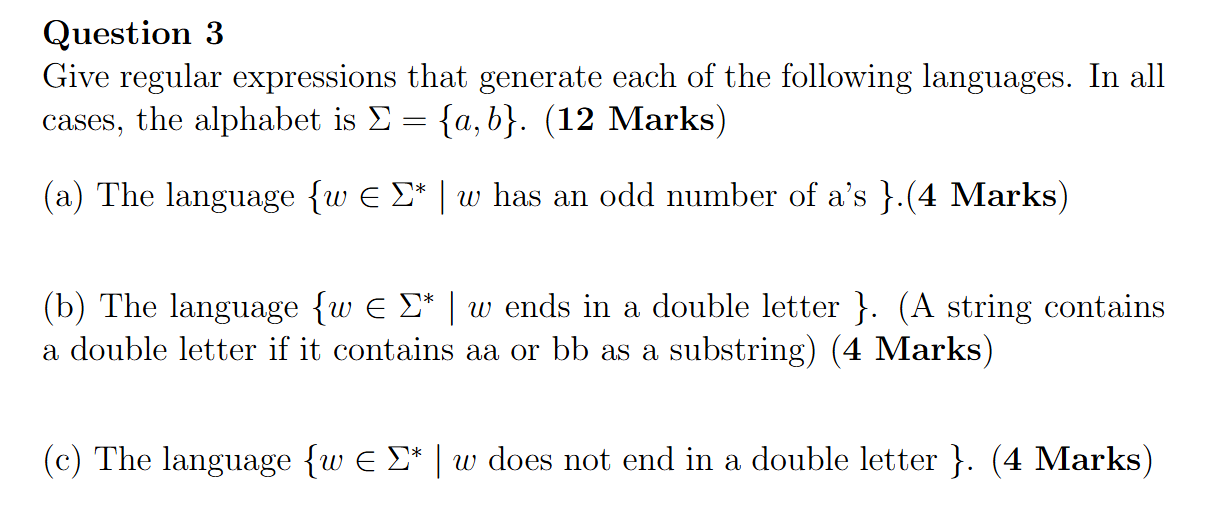
(a|b)\*ba

解释：

- `(a|b)\*` 表示 'a' 或 'b' 可以出现任意次，包括0次。

- `ba` 是字符串的固定结尾部分。

因此，这个正则表达式描述了所有可能的字符串，它们可以由任意数量的 'a' 和 'b' 开始，并以 "ba" 结尾。这正好符合题目中语言A的定义。



# Question 3 的答案如下：

Question 3

(a)

解释：

要匹配包含奇数个 'a' 的字符串，我们可以考虑以下模式：

- 奇数个 'a' 可以表示为a后跟偶数个 'a'。

- 偶数个 'a' 可以表示为(aa)\*，即 'aa' 的任意次重复。

因此，整个字符串可以由以下部分组成：

1. 任意数量的 'b'。

2. 一个 'a'。

3. 任意数量的 'b'。

4. 偶数个 'a'（即 (aa)\*）。

5. 任意数量的 'b'。

正则表达式：

**b\*(ab\*(aa)\*b\*)\***

(b) 语言

解释

要匹配以双重字母（即 'aa' 或 'bb'）结尾的字符串，我们可以考虑以下模式：

- 任意数量的 'a' 或 'b'。

- 结尾是 'aa' 或 'bb'。

正则表达式：

**(a|b)\*(aa|bb)**

(c)

解释：

要匹配不以双重字母（即 'aa' 或 'bb'）结尾的字符串，我们可以考虑以下模式：

- 任意数量的 'a' 或 'b'。

- 结尾是 'a' 或 'b'，但不能是 'aa' 或 'bb'。

我们可以通过排除法来构造这个正则表达式：

- 首先匹配任意数量的 'a' 或 'b'。

- 然后确保最后一个字符是 'a' 或 'b'，并且前一个字符不能与最后一个字符相同。

正则表达式：

**(a|b)\*(a(b|ε)|b(a|ε))**

详细解释：

- `(a|b)\*` 表示任意数量的 'a' 或 'b'。

- `(a(b|ε)|b(a|ε))` 确保最后一个字符是 'a' 或 'b'，并且前一个字符不能与最后一个字符相同：

- `a(b|ε)` 表示以 'a' 结尾，前一个字符可以是 'b' 或空。

- `b(a|ε)` 表示以 'b' 结尾，前一个字符可以是 'a' 或空。

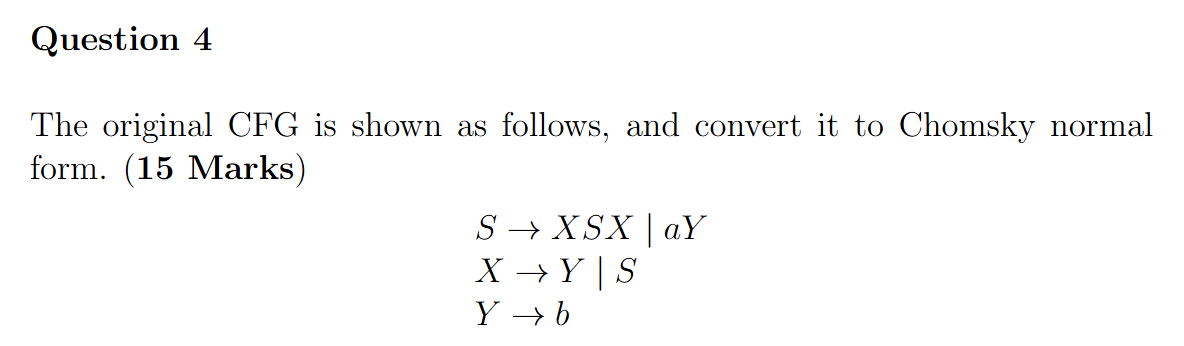
这样，我们确保了字符串不以 'aa' 或 'bb' 结尾。

总结

(a) b\*(ab\*(aa)\*b\*)\*

(b) (a|b)\*(aa|bb)

(c) (a|b)\*(a(b|ε)|b(a|ε))



# Question 4 的答案如下：

题目4要求我们将给定的上下文无关文法（CFG）转换为乔姆斯基范式（Chomsky Normal Form，CNF）。乔姆斯基范式是一种特殊的上下文无关文法，其产生式仅限于以下两种形式：

1. A → BC，其中A、B、C是非终结符。

2. A → a，其中A是非终结符，a是终结符。

下面是转换过程：

原始CFG：

S → XSX | aY

X → Y | S

Y → b

步骤1：消除单位产生式（Unit Productions）。

单位产生式是形如A → B的产生式，其中A和B是非终结符。在这个例子中，X → Y和X → S是单位产生式。我们可以直接删除这些单位产生式，因为它们不会影响生成的语言。

删除单位产生式后，CFG变为：

S → XSX | aY

X → b | S

步骤2：消除ε-产生式（Epsilon Productions）。

ε-产生式是形如A → ε的产生式，其中A是非终结符，ε表示空字符串。在这个例子中，没有ε-产生式，所以我们可以跳过这一步。

步骤3：将产生式转换为CNF形式。

我们需要将所有产生式转换为A → BC或A → a的形式。在这个例子中，S → XSX是复杂的，因为它涉及到三个非终结符。我们需要引入新的非终结符来分解这个产生式。

引入新的非终结符U和V，使产生式S → XSX变为以下两个产生式：

S → UV

U → XS

V → SX

现在我们的CFG变为：

S → UV | aY

X → b | S

U → XS

V → SX

接下来，我们需要确保所有的产生式都是CNF形式。注意到U → XS和V → SX仍然不是CNF形式，因为它们涉及到多个非终结符。我们再次引入新的非终结符来分解这些产生式。

对于U → XS，我们可以引入新的非终结符W，得到以下产生式：

U → XW

W → SX

对于V → SX，我们可以引入新的非终结符W'，得到以下产生式：

V → SW'

W' → X

现在我们的CFG变为：

S → UV | aY

X → b | S

U → XW

V → SW'

W → SX

W' → X

最后，我们需要检查是否所有的产生式都是CNF形式。在这个例子中，所有的产生式都已经符合CNF的要求。

最终，转换后的CFG（乔姆斯基范式）为：

S → UV | aY

X → b | S

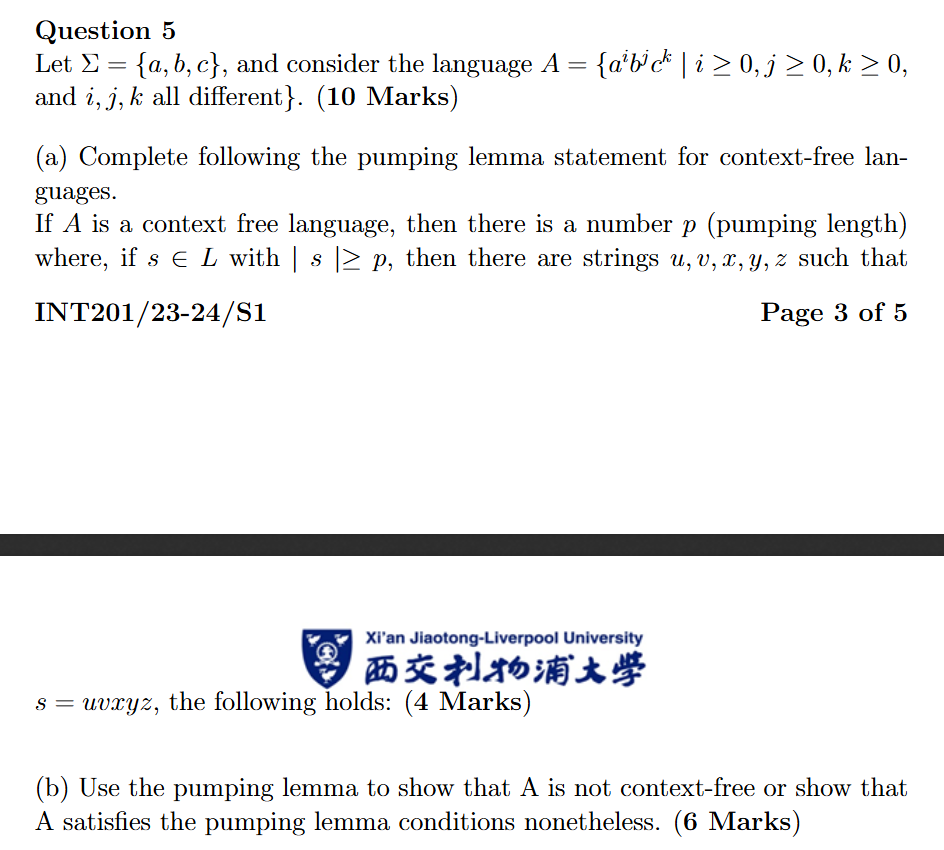
U → XW

V → SW'

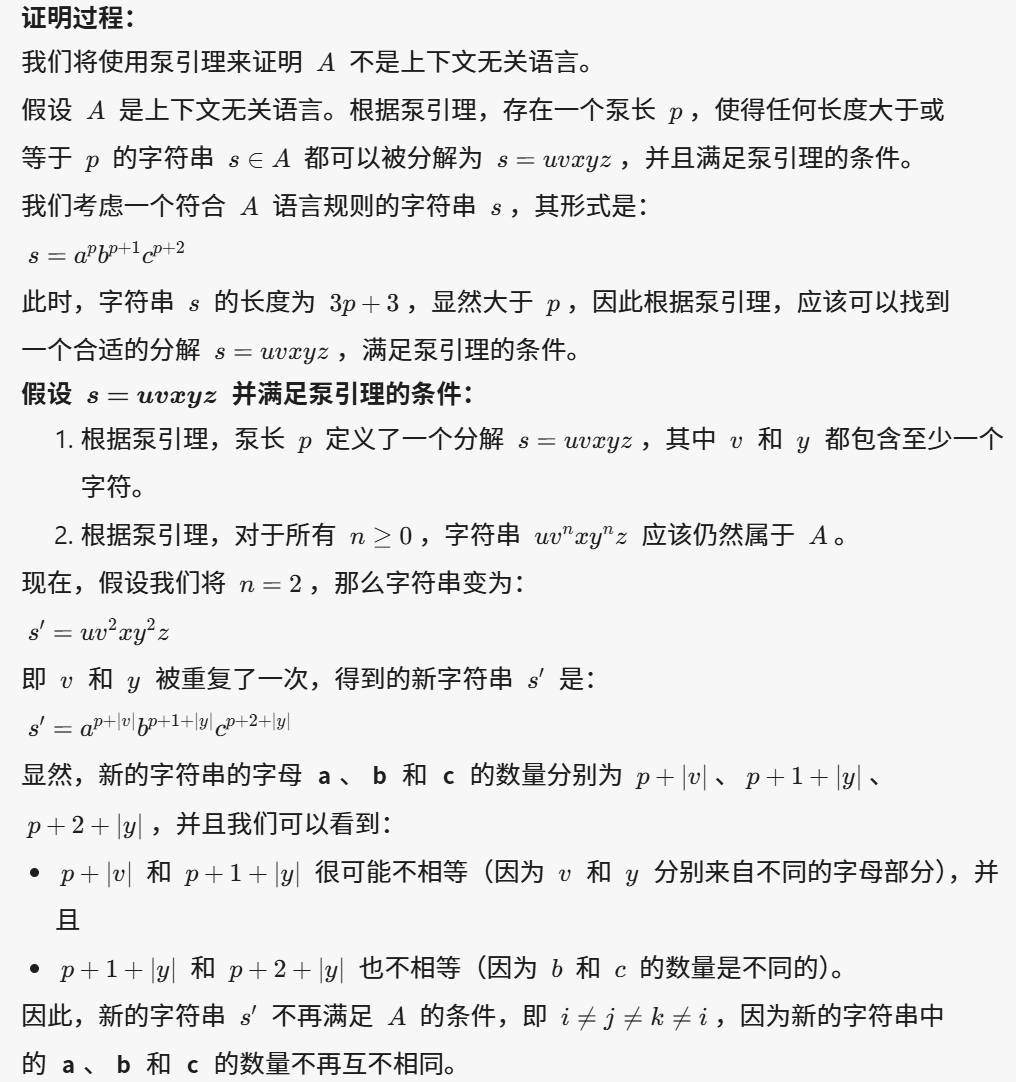
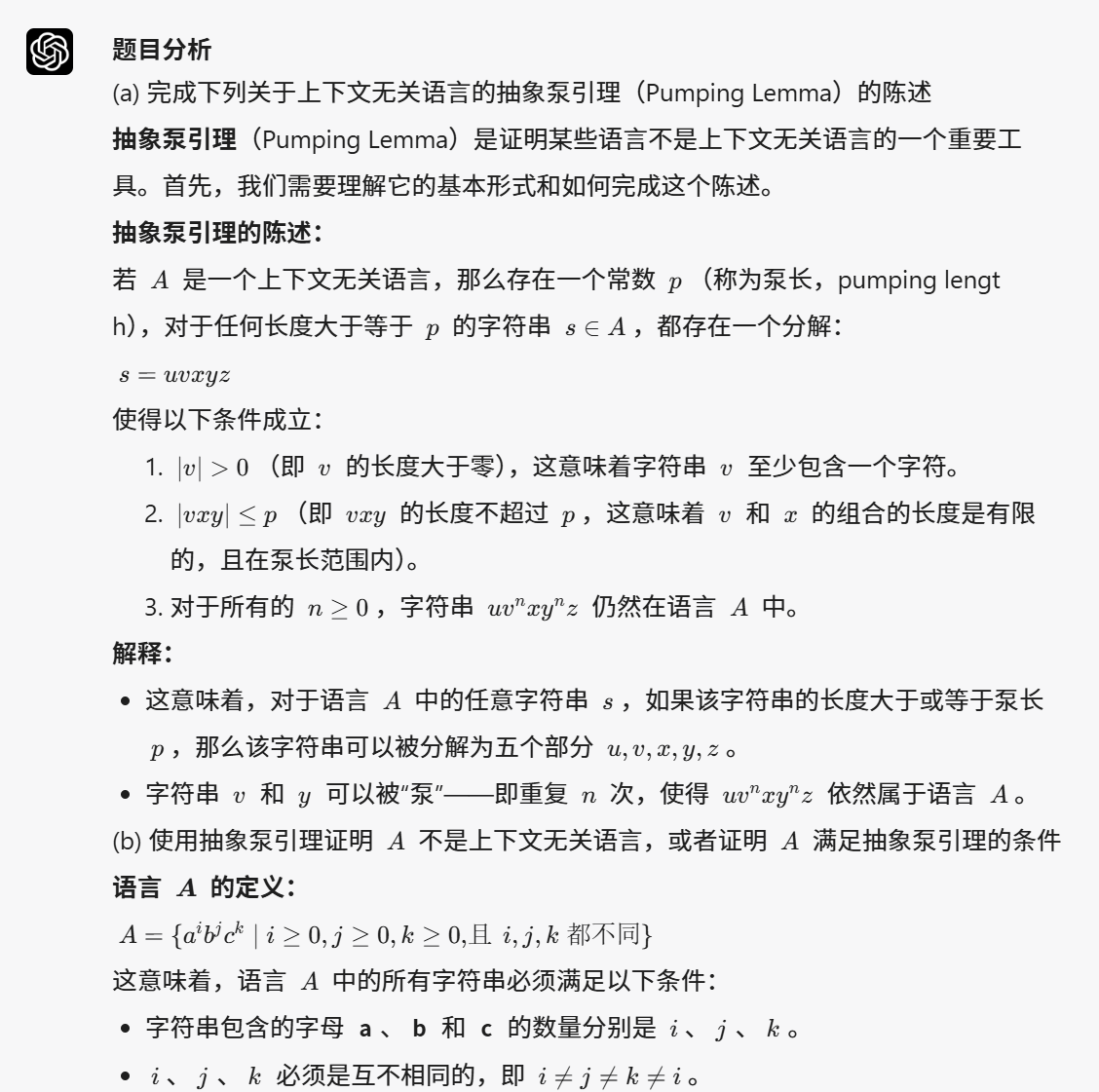
W → SX

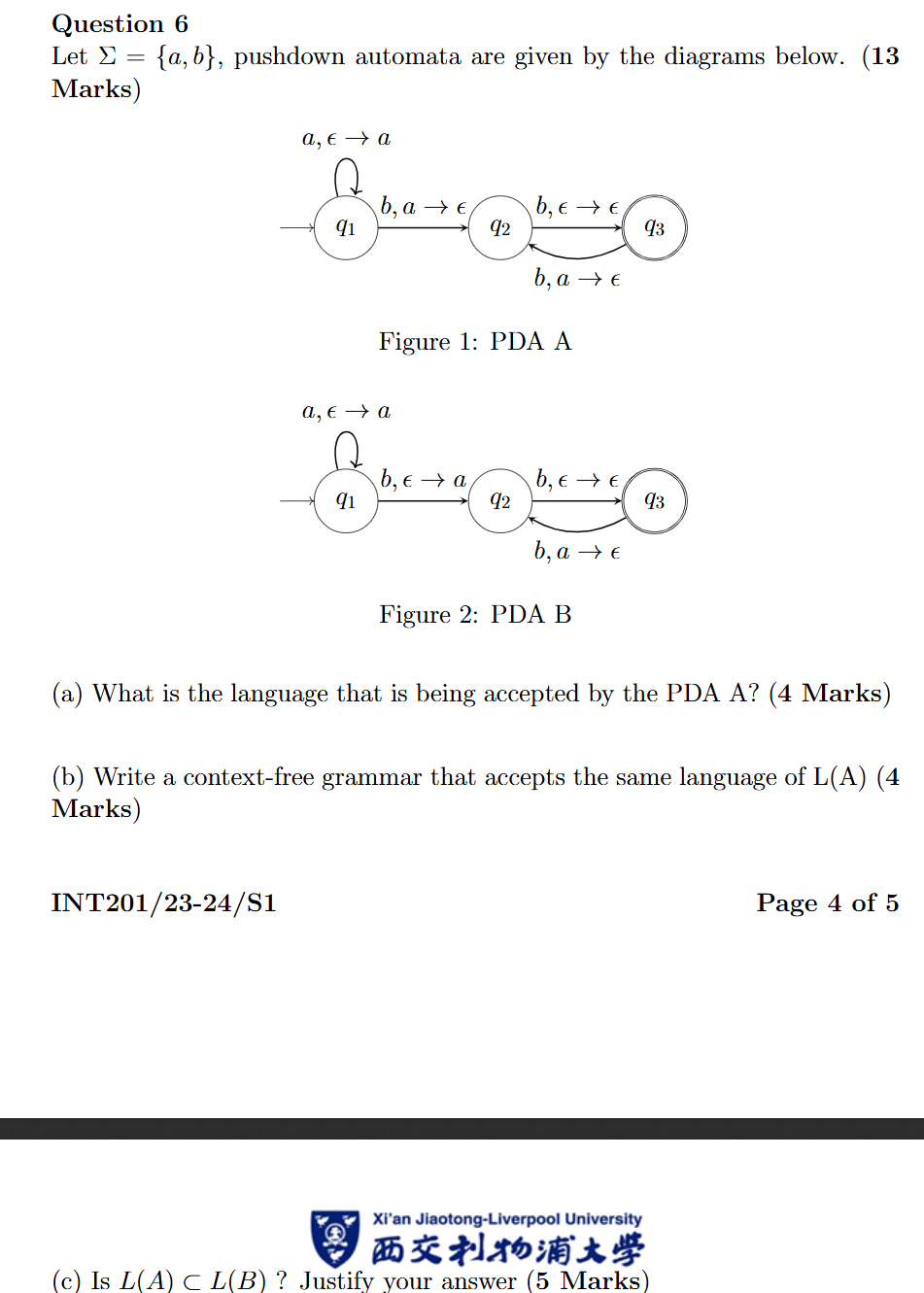
W' → X

这就是将原始CFG转换为CNF的详细过程。



# Question 5(第二问PPT L8 p21-24)





# Question 6

(a) PDA A 接受的语言是什么？（4分）

- 初始状态：`q1`

- 接受状态：`q3`

- 输入字符集：`Σ = {a, b}`

状态转移：

1. 在 `q1` 状态时：

- 输入 `a` 以及栈顶为空，将 `a` 压入栈，仍在 `q1` 状态。

- 输入 `b` 以及栈顶为 `a`，弹出 `a`，转换到 `q2` 状态。

2. 在 `q2` 状态时：

- 输入 `b` 以及栈顶为空，直接转换到 `q3` 状态。

- 输入 `b` 以及栈顶为 `a`，弹出 `a`，转换回 `q2` 状态。

由此我们可以看出，PDA A 接受的语言是所有形如 `a^n b^n (n ≥ 0)` 的串，即每有一个 `a` 就必须对应一个 `b`，且 `a` 在前，`b` 在后。因此，

答案 (a)：PDA A 接受的语言是 L(A) = `{a^n b^n | n ≥ 0}`。

(b) 写一个上下文无关文法，使其接受与 L(A) 相同的语言（4分）

上下文无关文法（Context-Free Grammar, CFG）的形式为 G = (V, Σ, R, S)：

- V 是变量集

- Σ 是终结符集

- R 是生成规则集

- S 是起始变量

为了生成 L(A) = `{a^n b^n | n ≥ 0}`，我们可以定义如下文法：

答案 (b)：

S -> ε | aSb

这个文法解释如下：

- `S -> ε` 表示空串，它属于 L(A)。

- `S -> aSb` 表示每有一个 `a` 就必须后跟一个 `b`。通过 S 的递归定义，我们保证了 `a` 和 `b` 数量相同且 `a` 总是在 `b` 之前。

(c) 判断 L(A) 是否是 L(B) 的子集并解释你的答案（5分）

首先我们需要分析 PDA B 的状态转移：

- 初始状态：`q1`

- 接受状态：`q3`

- 输入字符集：`Σ = {a, b}`

状态转移：

1. 在 `q1` 状态时：

- 输入 `a` 以及栈顶为空，将 `a` 压入栈，仍在 `q1` 状态。

- 输入 `b` 以及栈顶为空，将 `a` 压入栈，转换到 `q2` 状态。

2. 在 `q2` 状态时：

- 输入 `b` 以及栈顶为空，直接转换到 `q3` 状态。

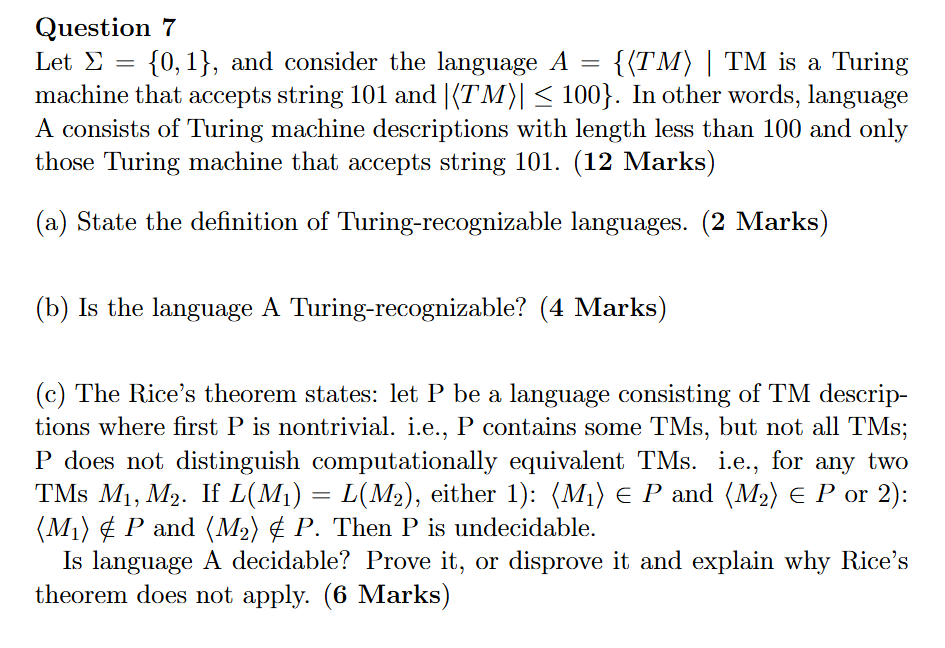
- 输入 `b` 以及栈顶为 `a`，弹出 `a`，仍在 `q2` 状态。

- 输入 `b` 以及栈顶为空，将 `a` 压入栈。

从 PDA B 的状态转移来看，状态`q1`和`q2`时输入 `b`，栈顶为空也可以并不停止接受新字符。这意味着 PDA B 可以接受更多样的字符串模式，包括形如 `a^m b^n (m, n ≥ 0)` 的串。而不仅仅是 `a` 和 `b` 数量相等。

答案 (c)：

L(A) = `{a^n b^n | n ≥ 0}` ⊆ L(B)，因为 PDA B 可以接受到符合 `a^n b^n` 模式在内的所有字符串，但同时也接受更多。因此，L(A) 是 L(B) 的子集。



(a) 陈述图灵可识别语言的定义。

图灵可识别语言（Turing-recognizable language）是指存在一个图灵机T，当给定一个字符串作为输入：

- 如果该字符串属于这个语言，则T最终会停机并接受该字符串。

- 如果该字符串不属于这个语言，则T要么会停机并拒绝该字符串，要么会不停机（即进入无限循环）。

换句话说，图灵可识别语言可以被图灵机接受但不一定能被图灵机决定。

**图灵可识别语言是指存在一个图灵机T，当给定一个字符串作为输入，如果该字符串属于这个语言，则T最终会停机并接受该字符串；如果该字符串不属于这个语言，则T要么停机并拒绝，要么不停机。**

(b) 语言A是图灵可识别的吗？（4分）

语言A的定义是包含所有接受字符串“101”的图灵机描述，且这些图灵机描述的长度不超过100。我们需要确定这样的语言是否是图灵可识别的。

为了证明语言A是图灵可识别的，我们可以设计一个图灵机T来识别它：

1. 图灵机T首先检查输入的图灵机描述是否符合语法且长度小于100。

2. 然后T模拟这个图灵机运行，给它输入字符串“101”。

3. 如果模拟器接受字符串“101”，则T接受这个图灵机描述；否则，T拒绝这个图灵机描述。

由于这样的图灵机T能够识别并接受所有符合条件的图灵机描述，语言A是可识别的。

答案 (b)：

**是的，语言A是图灵可识别的。我们可以设计一个图灵机来验证机器描述的长度以及它是否接受字符串“101”。**

(c) 语言A是可判定的吗？证明或反驳，并解释为何雷斯（Rice）的定理不适用。（6分）

雷斯（Rice）的定理指出：对于某个非平凡的语言P，其包含一些图灵机描述但不包含所有图灵机描述，并且P不区分计算等价的图灵机描述，那么P是不可判定的。

应用Rice定理需要检查两个条件：

1. P是否非平凡。

2. P是否不区分计算等价的图灵机描述。

对于语言A：

- A是非平凡的，因为确实存在接受字符串“101”的图灵机（这些图灵机的描述存在且长度小于100），但并不是所有的图灵机都能构造出这样的描述。

- 不过，语言A的描述实际包括了对图灵机描述长度的限制，而Rice定理只适用于不区分长度的计算等价性。因此，对于语言A，Rice的定理并不适用。

要判断语言A是否可判定，可以设计一个决策过程，即：

1. 检查输入的图灵机描述是否长度小于100，如果超长则立即拒绝。

2. 模拟图灵机，检查其是否接受字符串“101”。

由于我们既可以验证描述长度，也可以模拟图灵机的执行步骤，判断是否接受字符串“101”，这些都有明确的算法，因此语言A是可判定的。

答案 (c)：

**语言A是可判定的。我们可以验证图灵机描述的长度，并模拟图灵机执行，判断其是否接受字符串“101”。Rice定理不适用，因为语言A考虑了图灵机描述长度，而Rice定理仅适用于忽略长度的计算等价性。**