Propositional Logic

Implementation of truth-table enumeration algorithm for deciding propositional entailment.

i.e., Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

```
def pl_true(sentence, model):
    if isinstance(sentence, str):
        return model.get(sentence, False)
    elif isinstance(sentence, tuple):
        operator = sentence[0]
        if operator == 'not':
            return not pl_true(sentence[1], model)
        elif operator == 'and':
            return all(pl_true(arg, model) for arg in sentence[1:])
        elif operator == 'or':
            return any(pl_true(arg, model) for arg in sentence[1:])
        elif operator == '=>':
            return (not pl_true(sentence[1], model)) or
pl_true(sentence[2], model)
        elif operator == '<=>':
            return pl_true(sentence[1], model) == pl_true(sentence[2],
model)
    return False


def tt_check_all(kb, alpha, symbols, model, true_models, all_symbols,
col_widths):
    if not symbols:
        kb_true = pl_true(kb, model)
        alpha_true = pl_true(alpha, model)
        row_values = [str(model.get(s, False)) for s in all_symbols] +
[str(kb_true), str(alpha_true), str(kb_true and alpha_true)]
        formatted_row = "| " + " | ".join(f"{val:^{col_widths[i]}}" for
i, val in enumerate(row_values)) + " |"

        if kb_true and alpha_true:
            print(f"\033[92m{formatted_row}\033[0m")  # green for
satisfying rows
            true_models.append(model.copy())
        else:
            print(formatted_row)
        return (not kb_true) or alpha_true
    else:
        P = symbols[0]
        rest = symbols[1:]
```

```
        model_true = model.copy()
        model_true[P] = True
        model_false = model.copy()
        model_false[P] = False
        return (tt_check_all(kb, alpha, rest, model_true, true_models,
all_symbols, col_widths) and
                tt_check_all(kb, alpha, rest, model_false, true_models,
all_symbols, col_widths))


def tt_entails(KB, alpha):
    symbols = set()
    def extract_symbols(sentence):
        if isinstance(sentence, str):
            symbols.add(sentence)
        elif isinstance(sentence, tuple):
            for arg in sentence[1:]:
                extract_symbols(arg)

    extract_symbols(KB)
    extract_symbols(alpha)
    symbols = list(sorted(symbols))  # sorted for consistent order

    header_values = symbols + ["KB", "alpha", "KB ∧ α"]
    col_widths = [max(len(str(s)), 5) for s in header_values]

    total_width = sum(col_widths) + 3 * len(col_widths) + 1
    border = "─" * total_width

    print("\nTruth Table:")
    print("┌" + "┬".join("─" * (col_widths[i] + 2) for i in
range(len(header_values))) + "┐")
    header_string = "│ " + " │ ".join(f"{val:^{col_widths[i]}}" for i,
val in enumerate(header_values)) + " │"
    print(header_string)
    print("├" + "┼".join("─" * (col_widths[i] + 2) for i in
range(len(header_values))) + "┤")

    true_models = []
    result = tt_check_all(KB, alpha, list(symbols), {}, true_models,
all_symbols=symbols, col_widths=col_widths)

    print("└" + "┴".join("─" * (col_widths[i] + 2) for i in
range(len(header_values))) + "┘")

    print("\nModels where KB and alpha are true:")
    if true_models:
        for model in true_models:
            print(model)
```

```
    else:
        print("None")

    return result


# Example
kb2 = ('and', ('or', 'A', 'B'), ('=>', 'B', 'C'))
alpha2 = ('or', 'A', 'C')
print(f"\nDoes KB entail alpha? {tt_entails(kb2, alpha2)}")
```

OUTPUT:

Truth Table:

A	B	C	KB	alpha	KB ∧ α
True	True	True	True	True	True
True	True	False	False	True	False
True	False	True	True	True	True
True	False	False	True	True	True
False	True	True	True	True	True
False	True	False	False	False	False
False	False	True	False	True	False
False	False	False	False	False	False

```
Models where KB and alpha are true:
{'A': True, 'B': True, 'C': True}
{'A': True, 'B': False, 'C': True}
{'A': True, 'B': False, 'C': False}
{'A': False, 'B': True, 'C': True}

Does KB entail alpha? True
```