第 14 章 例外

> テキストの解答要約はこんな感じで引用表現にする（引用じゃないけど）

演習 14.1.1. [⋆]

Q. `error` が使われる文脈ごとに型を注記しては？

A. `error` を明示的には書かないが、`error` に簡約したいをいうケースがありうる。例えば、現在のバリアント（11章）においては、プログラマに対し `case` 式におけるバリアントのパターンを網羅的に書くことを要求している。一方、網羅的に書かなくても良いという拡張を行うと、プログラマが不要だと思ったパターンの文だけ記述量を削減できる。もし、実行時に網羅されていないパターンに該当する項が出現すると、`case` 式による評価は進行せず、行き詰まり状態になってしまう（進行則が成り立たなくなる）。そこで、記述されていないパターンが出現した場合は `error` を送出するという拡張に変更すれば、進行則は保たれる可能性がある(もちろん未検証）。

> 事態はもっと深刻で、型保存則が壊れる。

- 定理 14.1.2 [進行]
- T_{exn} を拡張可能バリアント型にするという話
  - 余談だけど Haskell には Extensible なるライブラリがあって無料で遊べちまうんだ！
  - ただ、型クラスがあるので実際のアプローチは (5) に近い

以下、解かずに参考文献を斜め読みするだけに留める。

演習 14.3.1. [⋆ ⋆ ⋆]

- 拡張可能バリアント型の説明を形式的にせよ
- 参考文献 The Definition of Standard ML
  - 多分、p.29 の規則 (30), (31) がそれ (Static Semantics)
    - Static Semantics と Dynamic Semantics があるようで、違いがよくわからない
    - というかよくわからない
  - exbind というのが新たに束縛される例外のレコードラベルを表している模様
  - VE は ValEnv（値環境??）

演習 14.3.2. [⋆ ⋆ ⋆⋆]

- 関数の型が送出しうる例外の集合を示すようにせよ。その体系が型安全であることを示せ。
- 参考文献 Type-Based Analysis of Uncaught Exceptions
  - Ext という型を用意している
  - 例外は副作用として扱っている？

演習 14.3.3. [⋆ ⋆ ⋆]

- Cont T に基づいた型付け規則の形式化をせよ。

- 参考文献 Typing First-Class Continuations in ML
    - フルペーパーなので頑張れば読めるかも...