Rust Chapter2

Why Iris?

K.Hirata

2.1 Separation logic

- Hoare logic の派生として 2001 年に誕生。pointer を扱う program をうまく扱うために作られた。

- **points-to assertion** $l \mapsto v$ という predicate があって、これは、location l に格納されている現在の値は v である、という knowledge を表し、現在のコードがその location を所有していることを表す。 (?)

- 所有権とは、書き換え可能権限のことで、プログラムの他の部分は l を書き換えて“干渉” できない、ということを表す。ただし、他の部分も読み取り権限などは持ちうる。

- Separation logic provides a way to make stable statements in an unstable world.

- 左随伴として **Separating conjunction** $P * Q$ というのもある。これは P と Q が、共に成り立つ、ヒープの別の部分に対しての predicate である、という意味の predicate になる。

- 例:

$$\textsc{frame}\ \frac{\{P\}\, e\, \{Q\}}{\{P * R\}\, e\, \{Q * R\}}$$

The frame rule says that any state not explicitly described in the precondition is unaffected by e.

To summarize, separation logic enables local reasoning about mutable state by (a) giving proofs a local way to talk about mutable state while excluding interference from other code, through ownership expressed by the points-to assertion; and (b) letting proofs preserve ownership of unrelated parts of the heap when applying specifications, through framing.

2.2 Concurrent separation logic (CSL)

- **並行計算**
  - CSL の key proof rule は、並行合成に関する次のルール

$$\textbf{PAR}\quad \frac{\{P_1\}\, e_1\, \{Q_1\} \qquad \{P_2\}\, e_2\, \{Q_2\}}{\{P_1 * P_2\}\, e_1 \parallel e_2\, \{Q_1 * Q_2\}}$$

  - $e_1 || e_2$ は 2 つの並行で計算されるスレッドで、独立に verify できることを表す。

  - これは frame の一般化として見れる。

$$\textbf{FRAME}\quad \frac{\{P\}\, e\, \{Q\}}{\{P * R\}\, e\, \{Q * R\}} \qquad \frac{\{P_1\}\, e_1\, \{Q_1\} \qquad \{R\}\, ()\, \{R\}}{\{P_1 * R\}\, e_1 \parallel ()\, \{Q_1 * R\}}$$

  - 逆に frame はもう一方のスレッドが何もしない並行計算だと見ることができる。

- # Resource invariants

  - 先ほどの PAR は、別々のスレッドが完全に独立している時の話で、もちろん **communicate** がある状況を考えないと意味がない。すなわち、何らかの状態を共有する場合も考えたい。

    - 何らかの状態: mutable heap, message passing channels

  - CSL はこの干渉を表現するために、”conditional critical region” に **resource invariants** を結びつける。

  - すなわち各 region には分離論理の論理式で表現される resource invariants が付随していて、この論理式は、何によって、その region が “guard“ もしくは “protect” されるか、を表す。

$$\text{CRITICAL-REGION}\quad \frac{\{P * I_r\}\, e\, \{Q * I_r\}}{\{P\}\ \texttt{with}\ r\ \texttt{do}\ e\ \texttt{endwith}\ \{Q\}}$$

- 例えば、critical region r に関する上のルールは、critical region r の中では I_r に対し、このスレッドはアクセスできるが、r が終わると I_r の所有権を返さないといけない、という意味になる。

- この手法では、一度 resource invariants が決定されると、各スレッドは、その invariants と、そのスレッドのすることだけが問題になる。

- すなわち他のスレッドの action は、invariants に abstracted away されているため、invariants 以外のことは気にしなくて良い。

2.3 Extensions of CSL

- Separation logic に PAR と CRITICAL-REGION を加えるだけで、プログラムから syntactic に自明でない形で共有されている資源の所有権について、エレガントに verify できることを O'Hearn が示した。

- CSL の発展の研究は主に次の2つの軸から行われる
  - 扱えるリソースの種類を (ヒープ以外に) 増やす
  - 表現できる protocol を増やす

More resources (CSL の扱えるリソースの種類を増やす)

- “Fictional separation” を扱いたい。すなわち、2 つのスレッドが物理的には同じ状態を触っていても、conceptual には状態に対して独立した logical part だと思いたい。(??)
  - 簡単な例は “fractional permissions” と呼ばれるもの。この論理では、points-to assertion は $l \mapsto^q a$ というように、有理数 q が付けられる。
  - そして、この points-to assertion は $l \mapsto^{q_1+q_2} v \quad \Leftrightarrow \quad \left(l \mapsto^{q_2} v\right) * \left(l \mapsto^{q_1} v\right)$ というように、separating conjunction で分割できる。例えば、q = 1 の時は full permission があって書き換え可能として解釈するが、q < 1 の時は読み取り専用になる、というモデルのように考えられる。
- この手法よって “user-defined resources” を証明中に resource の一つとして使うことができる (?)
- Iris で特に重要なのは Views Framework で、”separation algebra” を与えると、そこから分離論理を作ることができる (??)

The culmination of this development are "user-defined resources", where the user of a logic gets to pick an arbitrary instance of some algebra which they can then use as their notion of "resource" for their proofs. Particularly noteworthy for the development of Iris is the *Views Framework*:[15] a general recipe for how to define a separation logic given some "separation algebra" as a notion of resources, and given some relation between these resources and the physical state of the program. Through this relation, resources are viewed as "views" onto the physical state. The key innovation of the Views Framework is its *action judgment* that defines when a view may be "updated" from one resource to another. This idea of "updating" resources makes them behave a lot like a piece of state that can be manipulated in the logic but does not have to exist in the real program. And indeed these user-defined resources can take the role of auxiliary variables[16] in Hoare logic, also sometimes called "ghost state".

The Views Framework requires fixing the notion of resources before defining the logic. *Fictional Separation Logic*[17] demonstrates an approach for "dynamically" adding resources during the proof.[18]

More protocols (CSL の扱えるリソースの種類を増やす)

- (普通の) Resource invariants は、複雑な並行計算のデータ構造を表現するには足りない。

- 一般に Protocol はある種の “states” と “transitions” と “guards” を持って、スレッド間での非対称性を表す。(スレッドごとに異なる役割を持つ、という意味での非対称性)。

- このあたりは歴史的に継続して発展してきたが、“決定版” がなく、問題になっていたらしい。実際新しいライブラリや並行処理が生み出されるごとに新しい分離論理が必要になっていたらしい。

- RustBelt の著者は、Iris は “決定版” になり得ると主張している。

2.4 Iris

- Resource と Protocol を共にリッチにするのではなく、所有権の概念さえリッチにすれば、protocol をエンコードできる！
  - *Monoids and invariants are all you need.*
- とはいえこれには嘘が混じっている

Monoid <= resource はある種の monoid 的なもの。つまりだいたい resource のこと

1. 単純な monoid では不十分
   - Ghost state というのが出てくる。さらに higher order ghost state というのが出てきて……
2. Iris は monoid と invariants だけ、というわけではない
   - なんか色々大変らしい……
   - 大変そうだし意味不明なので割愛。
   - We refer the interested reader to “Iris from the ground up”, which explains in great detail how to construct a program logic with support for invariants using the base logic, and how to justify the soundness of the Iris base logic in the first place.