OBJECT SPACES

Motivation

- Integration von Interpreter (Code-Ausführung) und Daten/Objekten
  - Alle Operationen erwarten/liefern "application objects"
  - insbesondere: is_true (nötig für bedingte Sprünge)

Operationen

- Python-induzierte Operationen: id, type, issubtype, iter, next, repr, str, len, hash, getattr, setattr, delattr, ..., call, is_, exception_match
- Komfort: eq_w (==space.is_true(space.eq(o1, o2)))
- Objekterzeugung: newbool, newtuple, newlist, ..., wrap
- Unwrapping: is_true, int_w, str_w,
  - Konvertierung in RPython-Objekte
- Konstanten: w_None, w_True, sys, builtin, ...

Standard Object Space

- RPython-Wrapper-Objekte für "normale" Python-Objekte
  - verwendet z.B. für PyPy auf (unter?) Python
- Methoden-Dispatching unterstützt Multi-Methoden

Trace Object Space

- Wrapper um beliebigen Object Space, zwecks Tracing von Operationen

FLOW OBJECT SPACE

- Ziel: Abstrakte Interpretation
- Funktionsweise ähnlich zu Tracer
- Ausführung von Methoden/Codeblöcken mit Platzhaltervariablen
  - Bestimmung/Beschreibung von Basic Blocks

THUNK OBJECT SPACE

- Verzögerte Berechnung aller Operationen
  - tatsächliche Berechnung erst, wenn Wert benötigt wird
- ST become:

Taint object space

- Ziel: tainted objects
  - Vermischung von tainted und untainted objects vermeiden
  - geheime Daten (Passwörter)
  - nicht vertrauenswürdige Daten (Nutzereingaben)

Transparent Proxies

- Delegierung von Objektmethoden auf Proxy, ohne Typ des Objekts zu ändern
  - L = tproxy(list, f)