ASM Simplification Rules

At each step, the ASM finds the left-most *ready subexpression* in the workspace

- An expression involving a ***primitive operator*** (eg "+") is *ready* if all its arguments are values
  - Expression is replaced with its result
- A ***let expression*** `let x : t = e in body` is *ready* if `e` is a value
  - A new binding for `x` to `e` is added at the end of the stack
  - let expression is replaced with `body` in the workspace
- A ***variable*** is always *ready*
  - The variable is replaced by its binding in the stack, searching from the most recent bindings
- A ***conditional expression*** `if e then e1 else e2` is *ready* if `e` is either `true` or `false`
  - The workspace is replaced with either `e1` (if `e` is `True`) or `e2` (if `e` is `False`)
- A **constructor expression** (record, tuple, defined type) is *ready* if all its arguments are values
  - Storage space is created in the heap
  - Expression is replaced with a pointer to the storage
- A **pattern match expression** is *ready* if the item to be matched is a value
  - Find the first pattern that matches and replace the expression with the corresponding result
- A **fun expression** is always *ready*
- A **function call** is *ready* if the function and all of its arguments are values
  - The current workspace is pushed to the stack with an open space for the return value
  - Function argument bindings are pushed to the stack
  - The function body is placed in the workspace
- If the workspace contains a single value, and there is unfinished work on the stack
  - Place the current workspace value into the most recent unfinished work on the stack in the space left open
  - Remove all bindings on the stack up to the unfinished work
  - Move the contents of the unfinished work back into the workspace