Some ideas on a proof technique language

Gudmund Grov (Edinburgh) & **Lucas Dixon** (Google)

Sponsored by: EPSRC funded AI4FM project, Bundy's platform grants & Google

Motivation

- Being able to write special-purpose reasoning techniques is a key to TP automation
- Writing and reading/understanding reasoning techniques is difficult
- In particular, how to treat and passing around goals and results
- Hard to debug and apply (static) analysis
- We motivate and introduce a few ideas for a language which addresses these issues
  - Simple/abstract representation to enable learning
  - We use IsaPlanner for this work

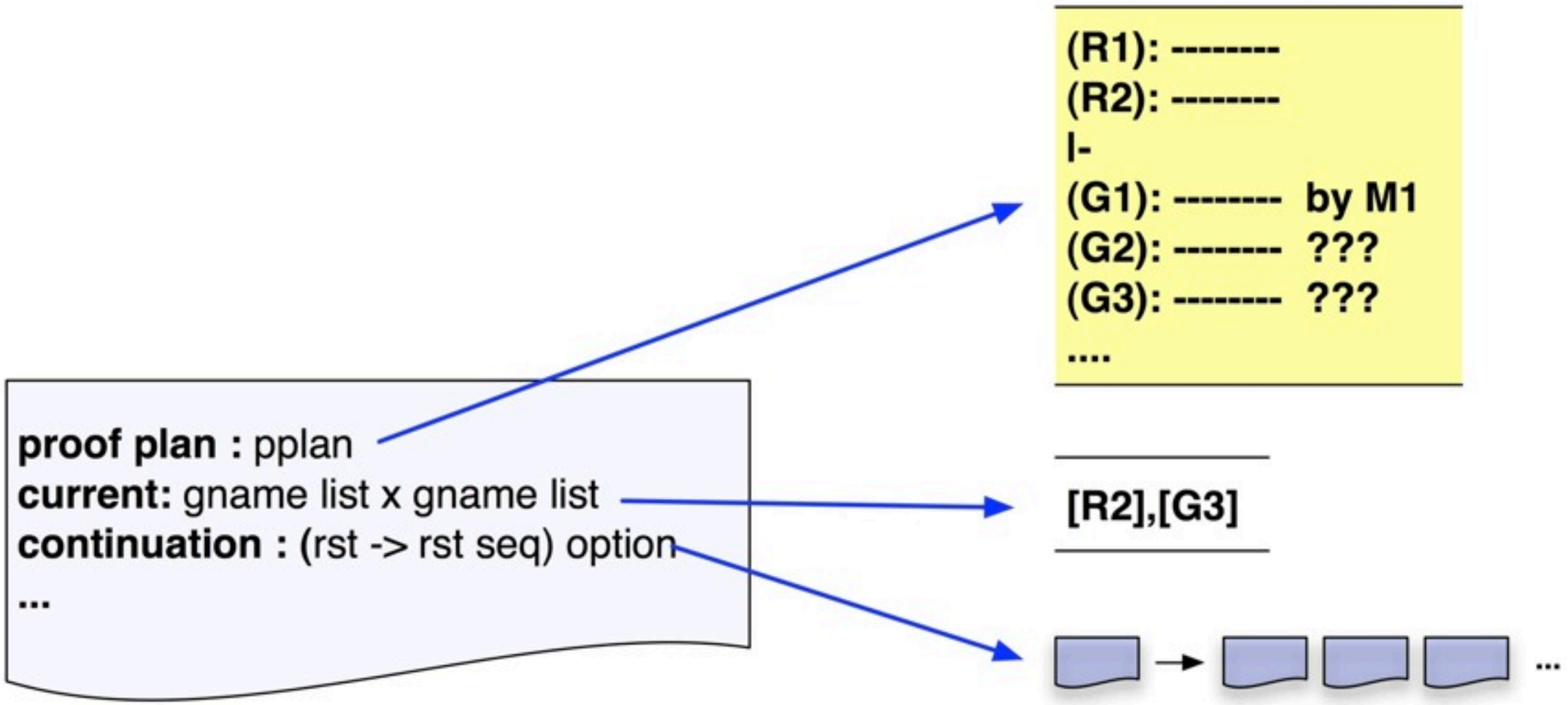
LCF Tactics

- List of goals - reached by number in list
- New goals typically added first in list
- Need to keep track of new goals etc
  - e.g. `apply ta 1; apply tb 2; apply tc 2;`
  - we don't know if tactics will solve goal or produce new ones
- Combinators gets some way
  - e.g. `apply ta 2;  ALL_NEW_GOALS tb`
- HiTac/HiProof: Hierarchical view of proofs
  - non-deterministic split/append of goal lists

IsaPlanner-2

- Reasoning states (`rst`) - includes (among other things)
  - a proof plan
    - all goals are named (e.g. `apply x to goal g`)
  - keeps a list of open/current goals and results
  - optional continuation (reasoning technique)
- Reasoning techniques (`rtechn`)
  - operations on reasoning states: `rst -> rst seq`
  - lazy application (unfolding of continuation)
  - separates search from the search space
  - interesting combinators: (lazy tree) map, fold, subspace

IsaPlanner-2



Problems

- No separation between `rtechn` and `rst -> rst seq`
- Goal handling - no easy way to refer to goals
  - ``current'' goals/results - typically new goals/result from rtechn
  - write a technique as a function on a goal (`goal -> rtechn`)
  - e.g. `(apply rt1 to g1) then (map_then rt2) then (set_cur [],[g1]) then (map_then rt3)`
- Static checking of combinations difficult, e.g.
  - wrong example: no IH when rippling or no assumption in fwd step
  - wrong goal: ripple a condition and not the goal

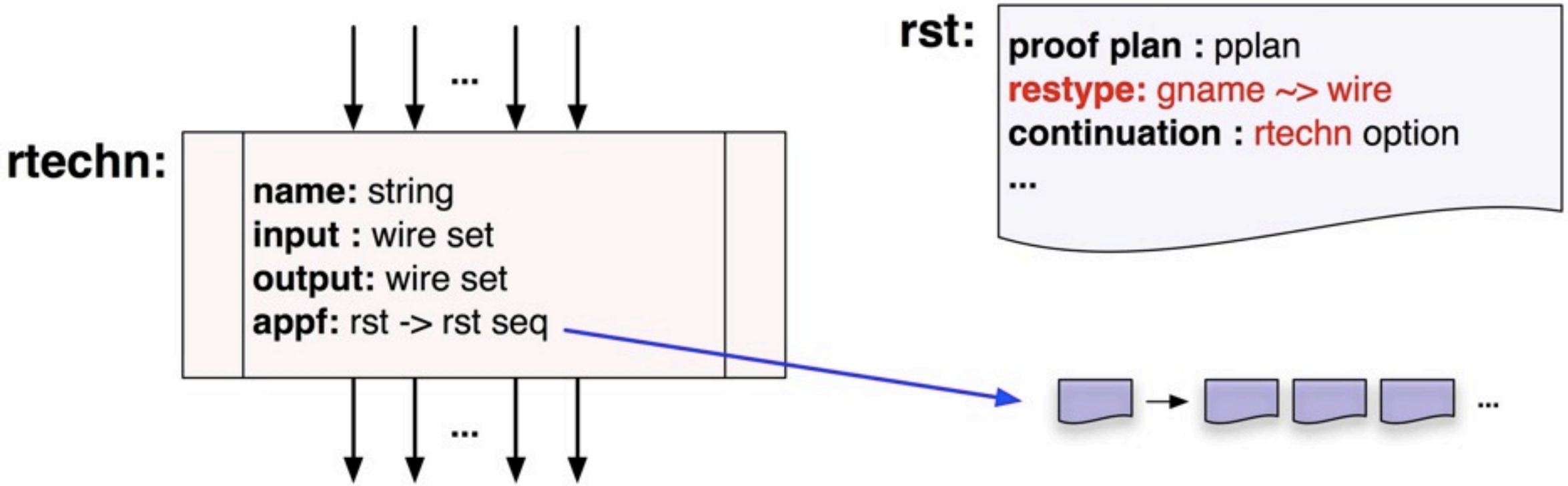
Requirements

- Clear & simple semantics
  - uniform handling of goals
- Static checking
  - correctness-by-construction
- Easy to read & write techniques
- Flexible approach
  - easy to change if we get it wrong...
- Abstract/simple ~ machine learnable (techniques)

IsaPlanner-3

- Boxes and wires
  - boxes are techniques
    - wires are goal/result types
      - some I/O wires may be "empty"
- Abstracts over actual goals/results
- `restype` map in `rst` for goals/results

IsaPlanner-3



Wires

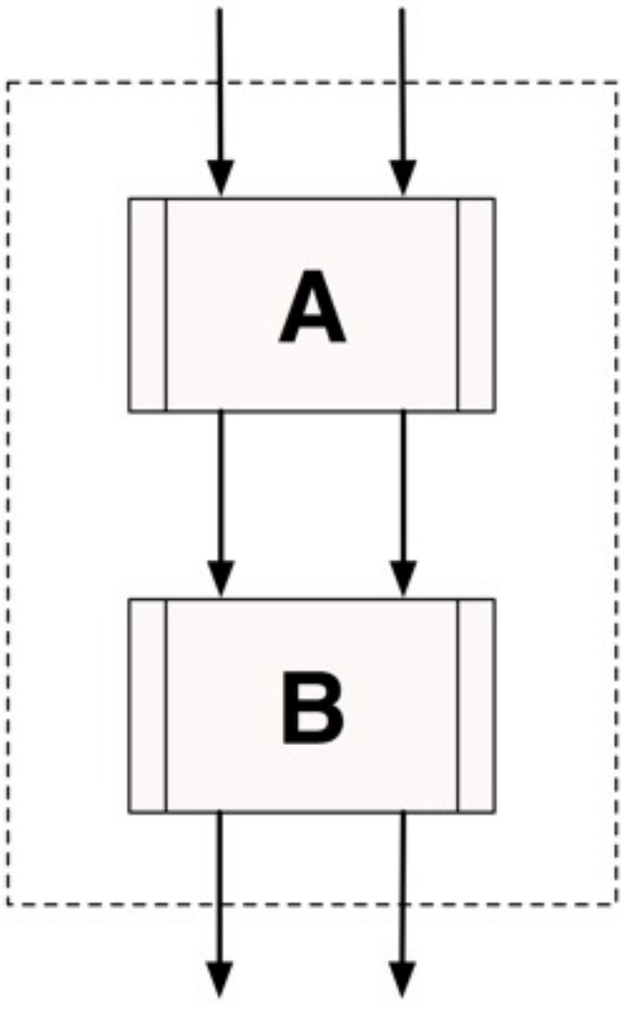
- A wire describes a "type of goal/result"
  - a nice way of classifying them
- Currently represented as strings
- Target has to be "more general" than source
  - partial order on wires as string with dot notation
    - e.g. "A.B" < "A"
- Separate BCK/FWD and AND/OR wires [more later]

FWD/BCK wires

- Techniques on goals are backwards
  - from goals to subgoals
  - linear: goal consumed & new goals created
  - input wire must be consumed
  - Q: what happens with discharged goals?
- Forward application from result to result
  - should be able to reason forwards from same result many times
  - input wire not consumed
- We separate forwards and backwards wires
  - "goal.x" vs "result.x"

Combinators

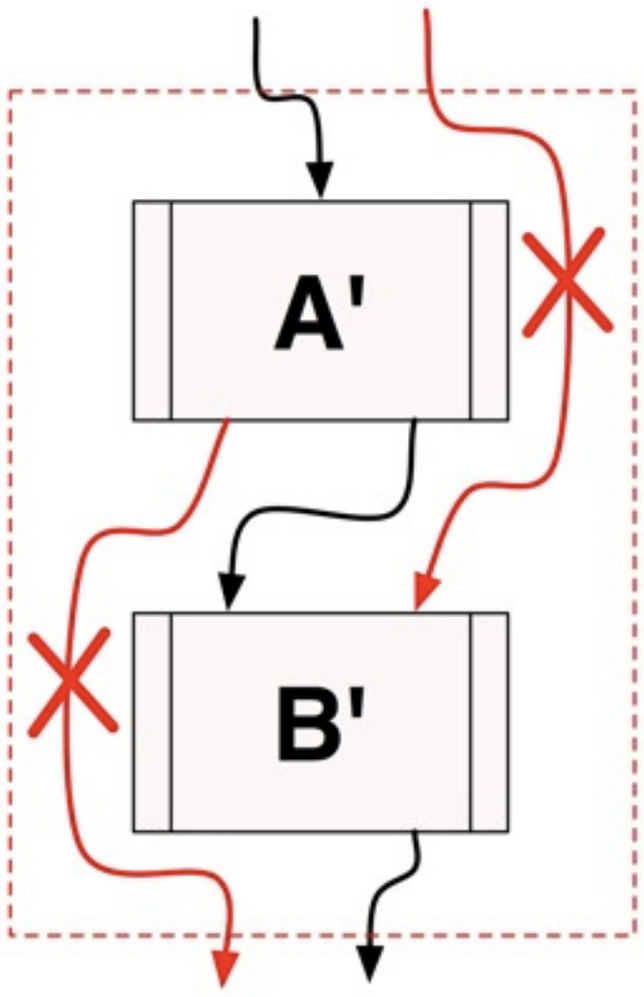
(A then B)



- Sequential composition
- I/O type ensured by combinator
  - `input(B) = output(A)` [almost]
  - `input(A then B) = input(A)`
  - `output(A then B) = output(B)`
- Composition/separation clear

Combinators

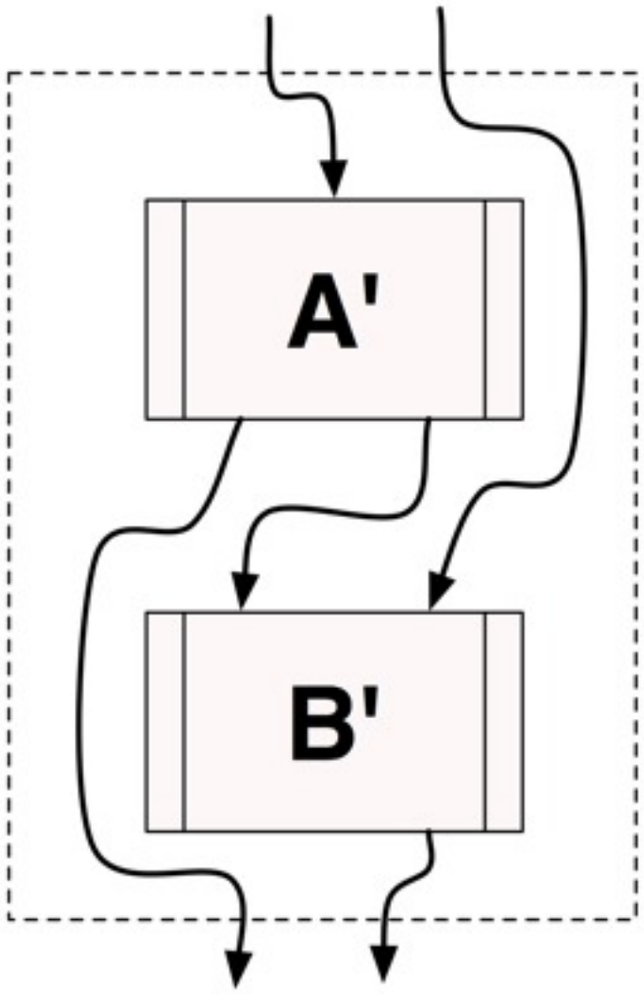
(A then B)



- Sequential composition
- I/O type ensured by combinator
  - input(B) = output(A) [almost]
  - input(A then B) = input(A)
  - output(A then B) = output(B)
- Composition/separation clear

Combinators

(A compose B)

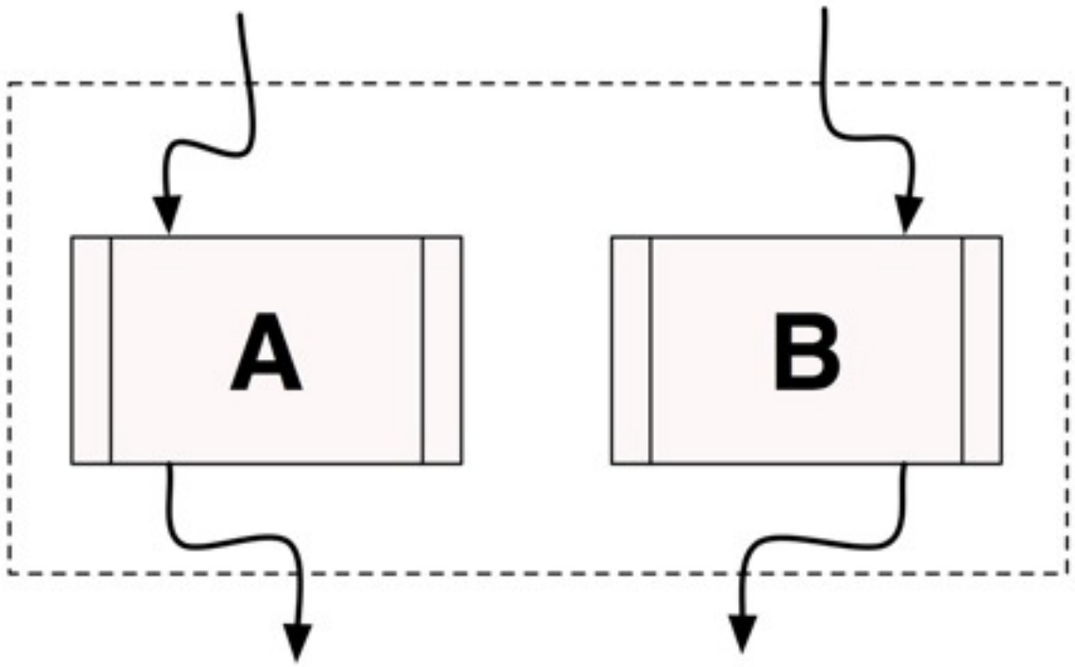


- Generalises then
- Allows bypassing of wires
- composition/separation less clear

- input(A' compose B') =
  input(A') + (input(B') - output(A'))
- output(A' compose B') =
  output(B') + (output(A') - input(B'))

Combinators

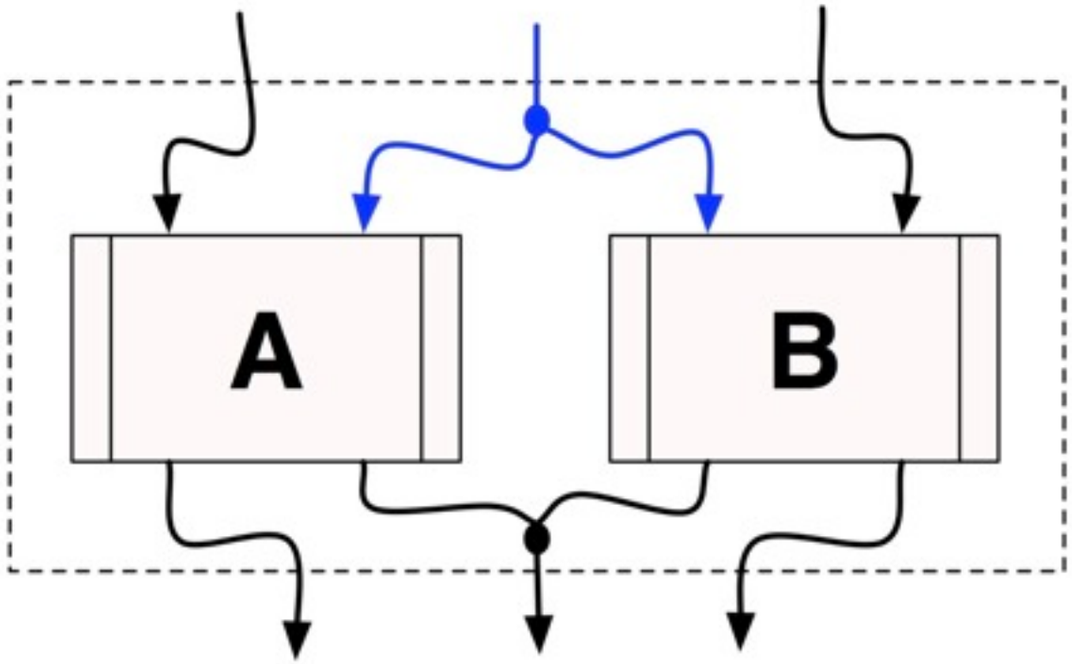
(A `tensor` B)



- Symmetric: `A tensor B = B tensor A`
  - can be parallelized
  - evaluation:
    - “run as in parallel (on same input) - combine results”

Combinators

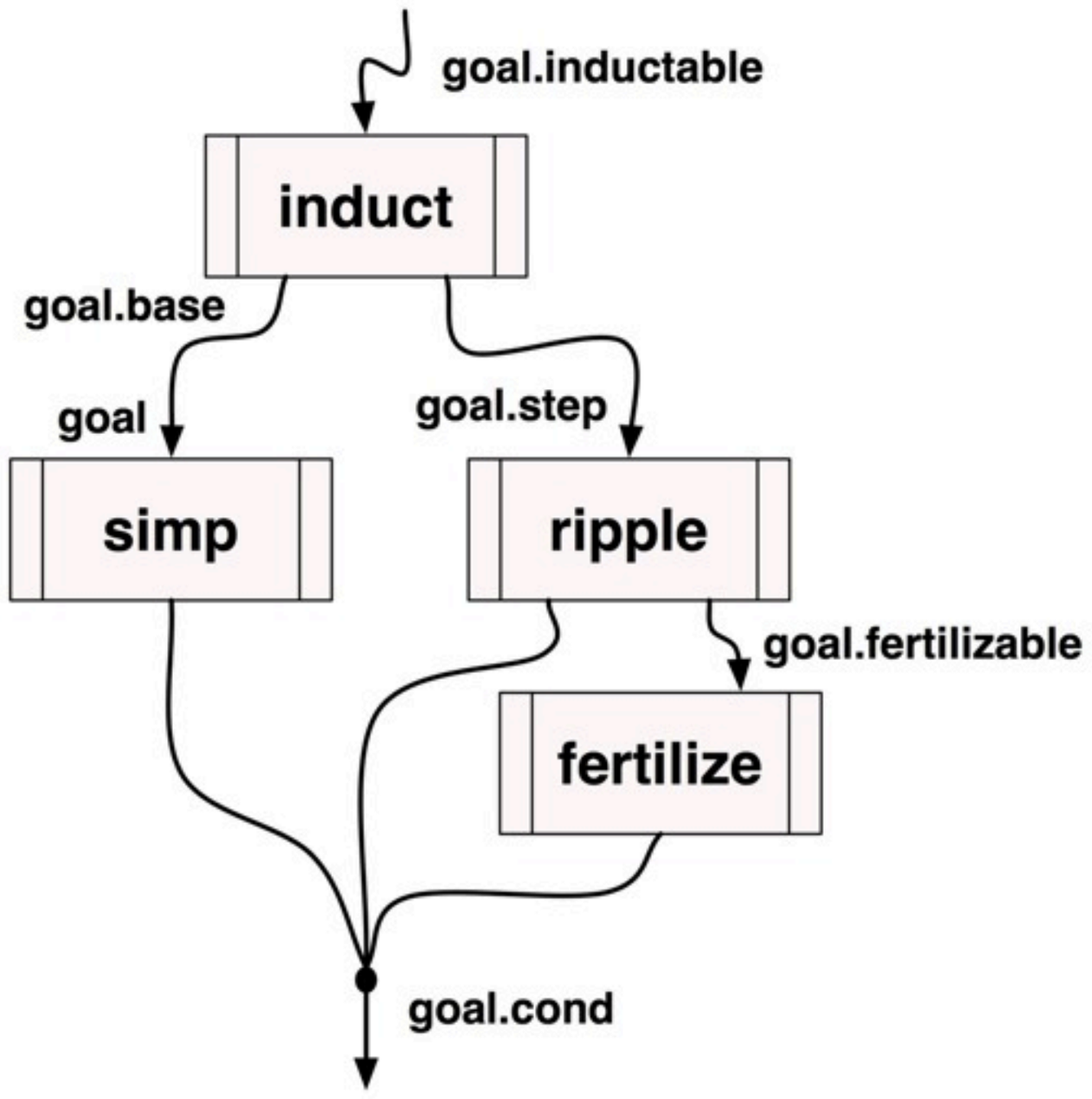
(A tensor B)



- Joint input (blue wire): only if fwd
  - or no meta variables
- Backward input wires must be disjoint
- no such requirement of output (due to eval)

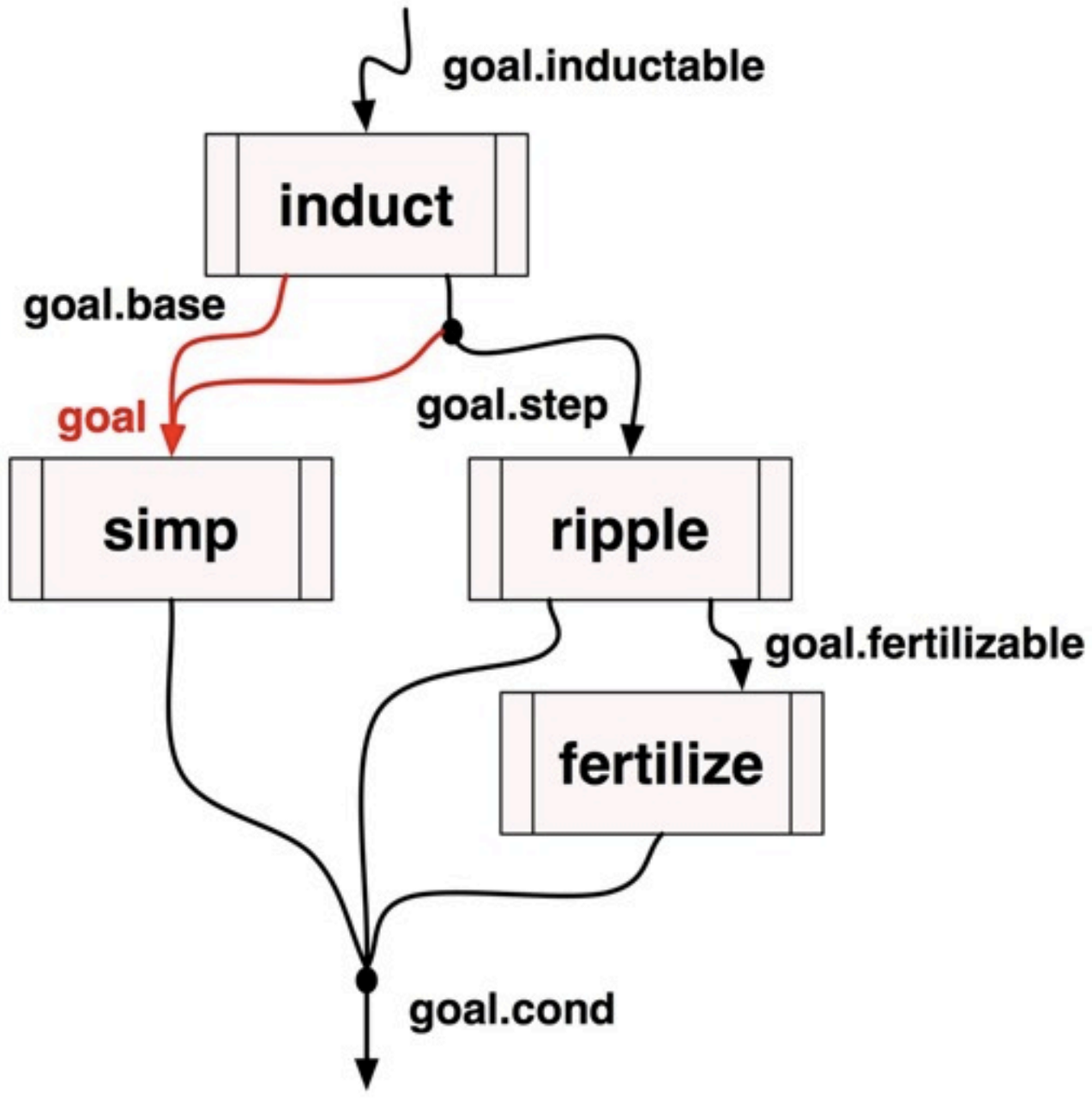
Example (BCK only)

induct then (**simp** tensor (**ripple** compose **fertilize**))



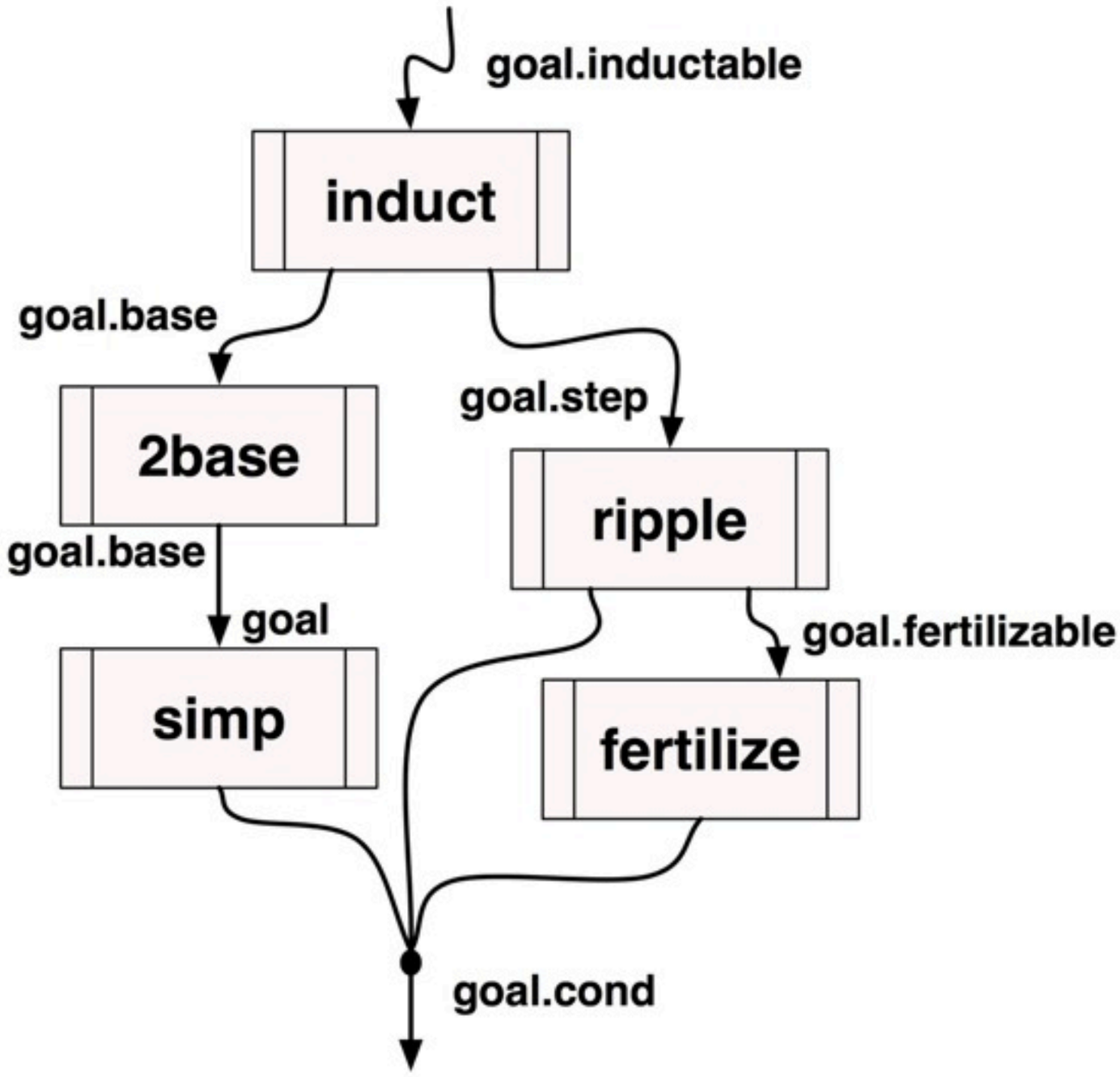
Example (BCK only)

induct then (**simp** tensor (**ripple** compose **fertilize**))



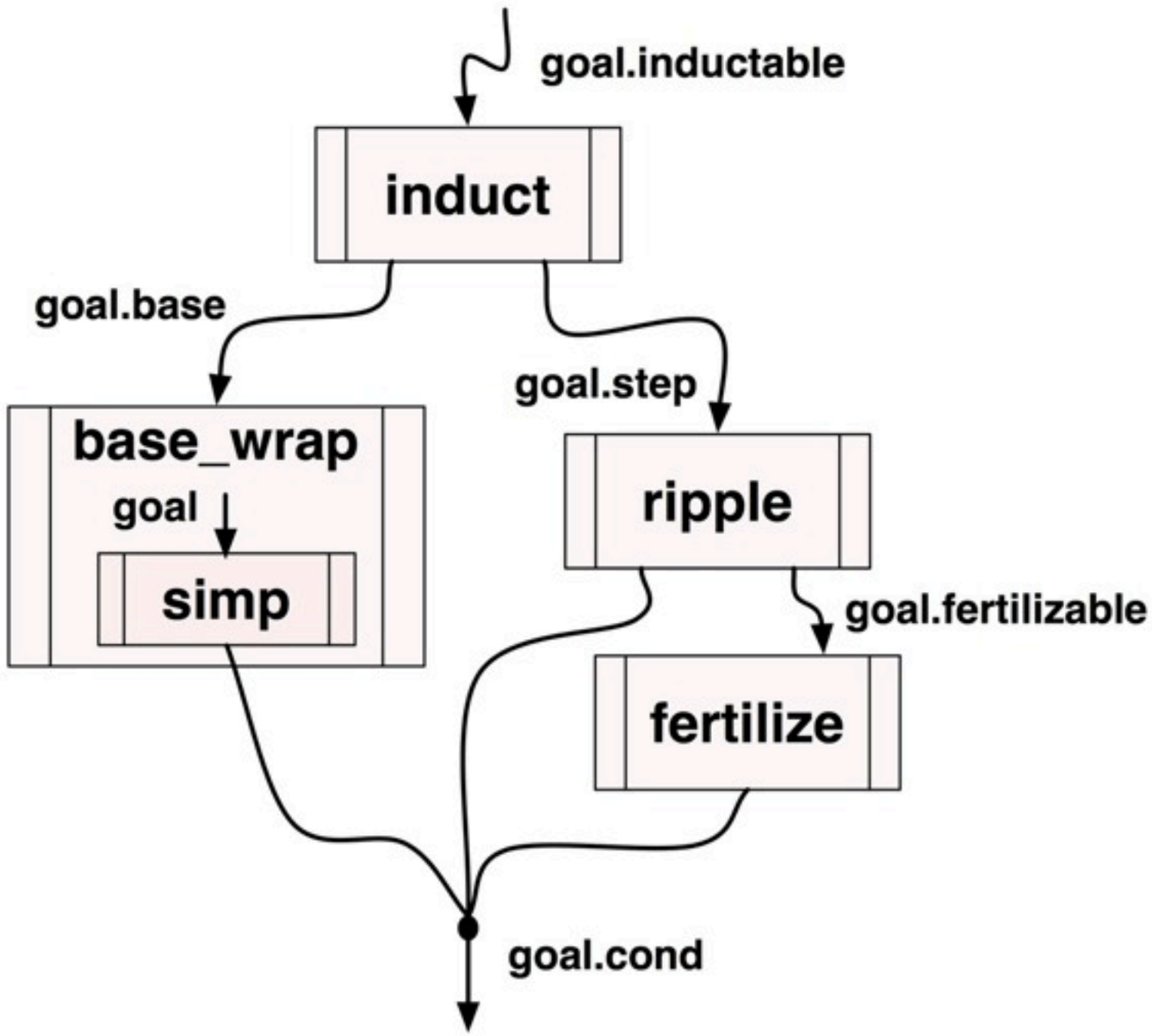
Example (BCK only)

induct then ((**2base** then **simp**)
tensor (**ripple** compose **fertilize**))

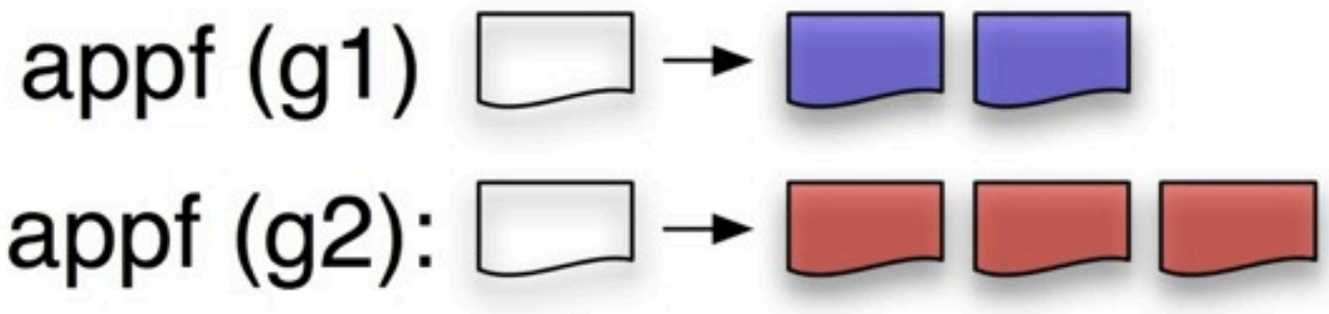


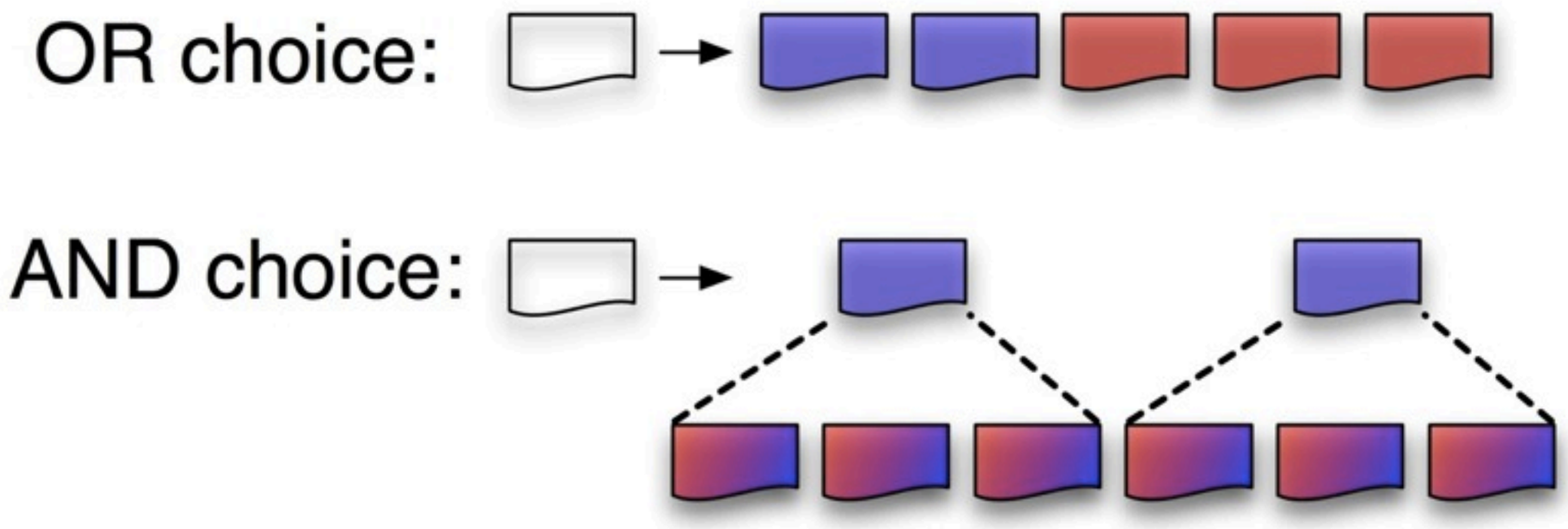
Example (BCK only)

induct then ((**base_wrap simp**)
tensor (**ripple** compose **fertilize**))



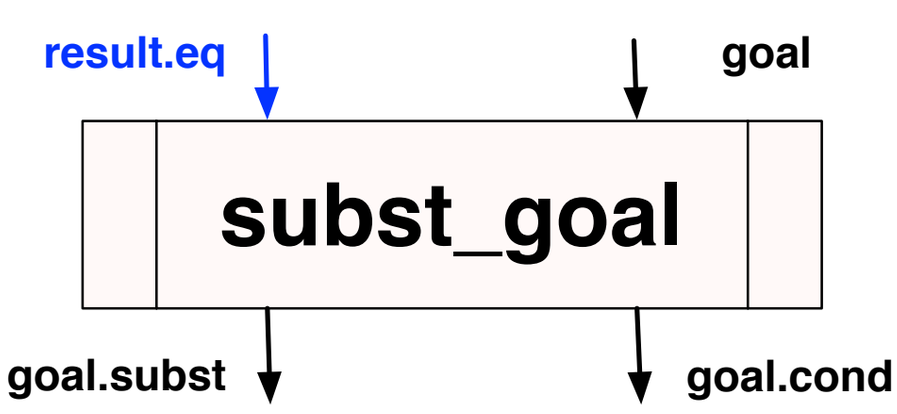
AND/OR wires





AND/OR wires

- No syntactic difference in in language between AND and OR
- In most cases wires are AND choices:
  - e.g. we simplify all base cases
- But, there are cases we want OR choices, e.g **substitution**:



- Problem: non-determinism - Example (with result.eq as AND)
  - **Input**: ({a=b,c=a, q => a=e},{P(c)})
  - **Output**: ({P(a)},{}) **or** ({P(b)},{}) **or** ({P(e)},{q}) **?**

Static checking

- Easier to read and write techniques
- Clear evaluation semantics
- Easier to debug
- Can we resort to ML type checking?

(Some) future work

- **Wires** - parameterize over them
  - keep wire/type-checking decidable & static
  - more structure than names for better classification (reg-expr/1st order)
  - introduce framing in `restype` map?
- **Application function (appf)** - still a "black box" (cannot decompose)
  - loops: only low-level repetition
  -
```
datatype appf = Comp of (rtechn * rtechn)
              | Tensor of (rtechn * rtechn) ...
```
  -
```
datatype appf = Nested of (rtechn HGraph)
              | Atom of rst -> rst seq
```
- **Wire classification/learning** - sufficiently simple/abstract language to
  - recognise patterns where techniques succeeds/fails
  - automate classification & re-classify (specialise) goals/results

AI4FM 2011

- If you want to escape from the Royal wedding...
- AI4FM 2011 in Edinburgh 28-29 April
  - see http://www.ai4fm.org/
- Please contact me if you are interested