Try hard!

Proof automation with monads

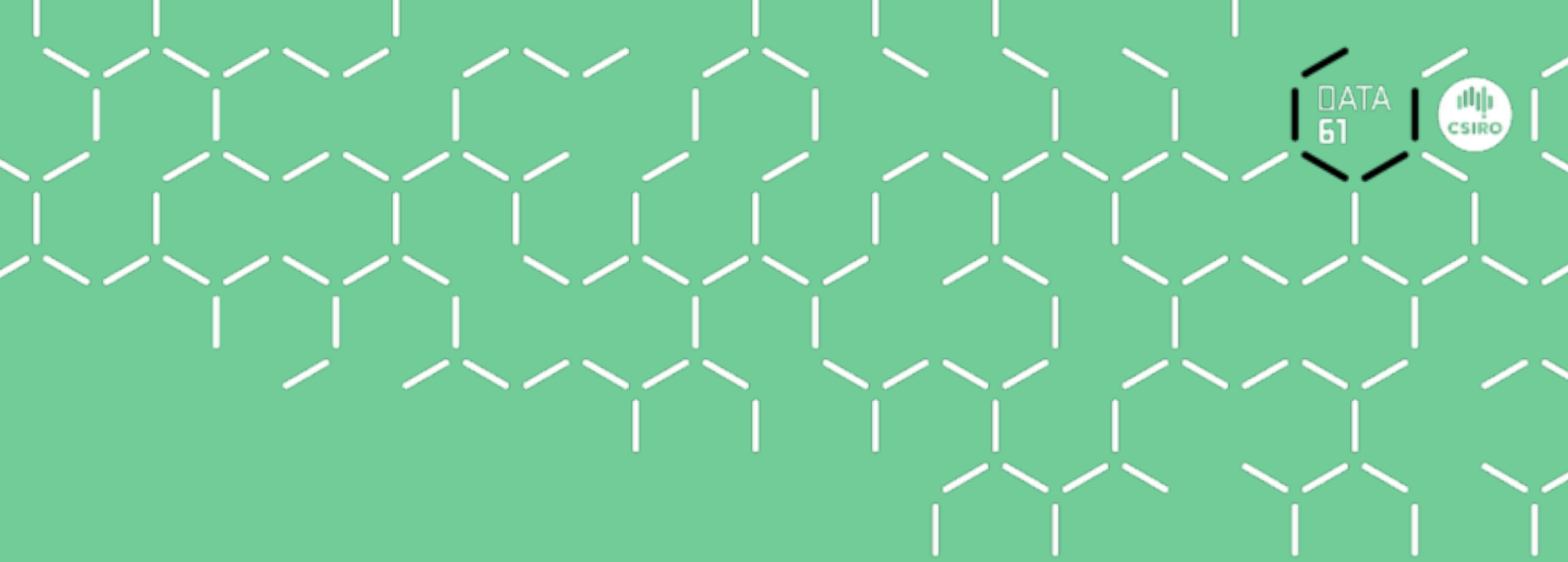
Yutaka Nagashima | Software Systems Research Group

The Sydney Area Programming Languages Interest Group (SAPLING) in November 2015

www.csiro.au

Conclusions first

- I am developing a proof automation tool for Isabelle/HOL.
- I am using monads for this.
- It can discharge some proof obligations that Isabelle's default automation tools cannot prove.



First Try:

Demo 1

Isabelle/HOL 101 in 3 minutes

Try the “try” command

Tactics 1

goal

tactic

Case 1

new goal

Case 2

no subgoal

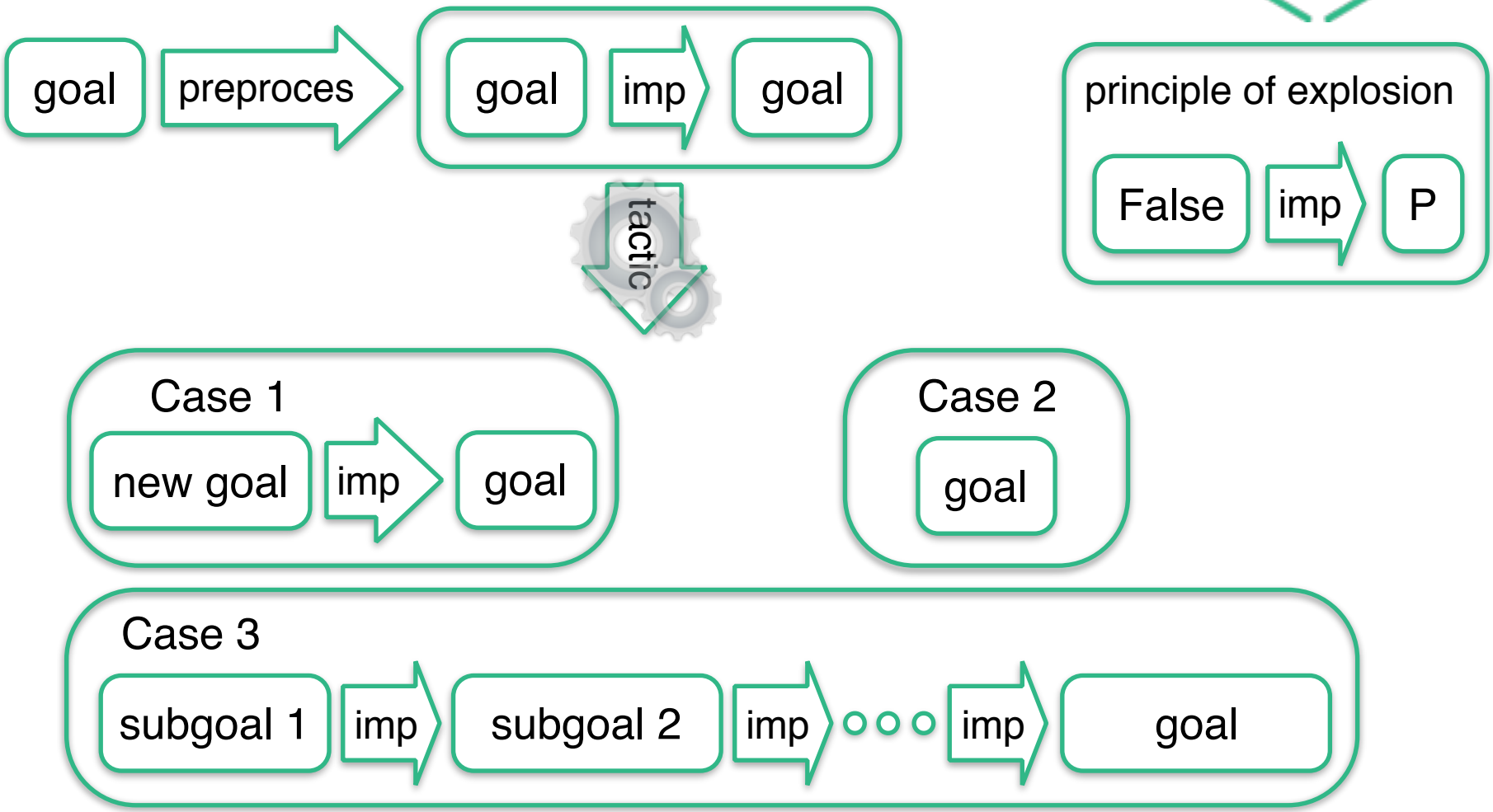
Case 3

subgoal 1 subgoal 2 ○ ○ ○ subgoal n

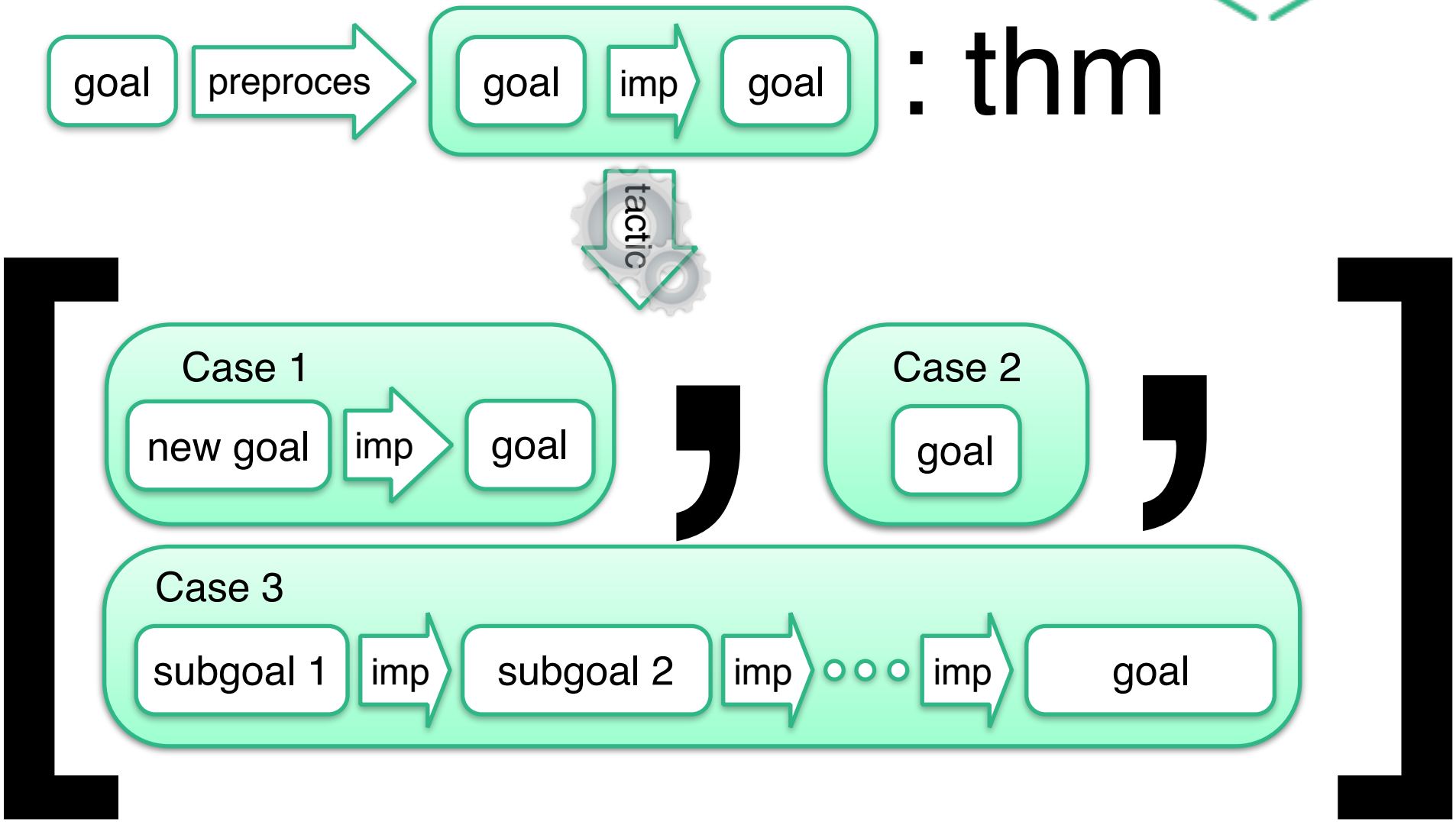
Case 4

the same goal with error message

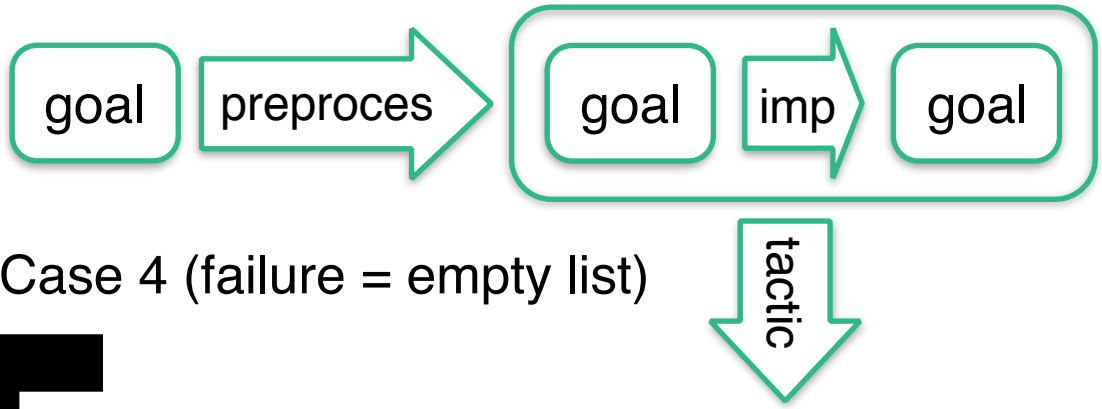
Tactics 2


goal
preproces
goal
imp
goal
tactic
principle of explosion
False
imp
P
Case 1
new goal
imp
goal
Case 2
goal
Case 3
subgoal 1
imp
subgoal 2
imp
imp
goal

Tactics 3


goal
preproces
goal
imp
goal
: thm
tactic
Case 1
new goal
imp
goal
,
Case 2
goal
,
Case 3
subgoal 1
imp
subgoal 2
imp
imp
goal

Tactics 3



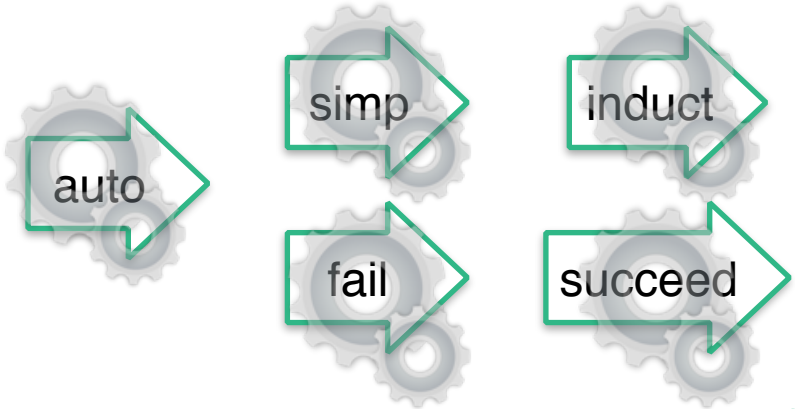
Case 4 (failure = empty list)

[ ]

Tactics 4

Lazy

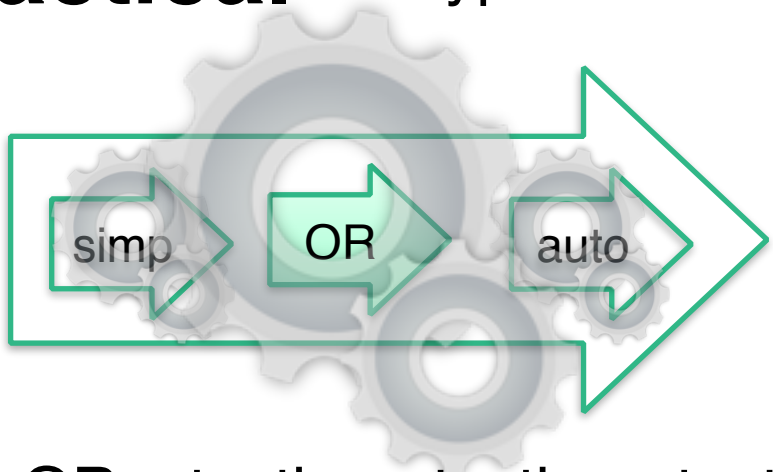
fun tactic :: thm -> [thm]
type tactic = thm -> [thm]



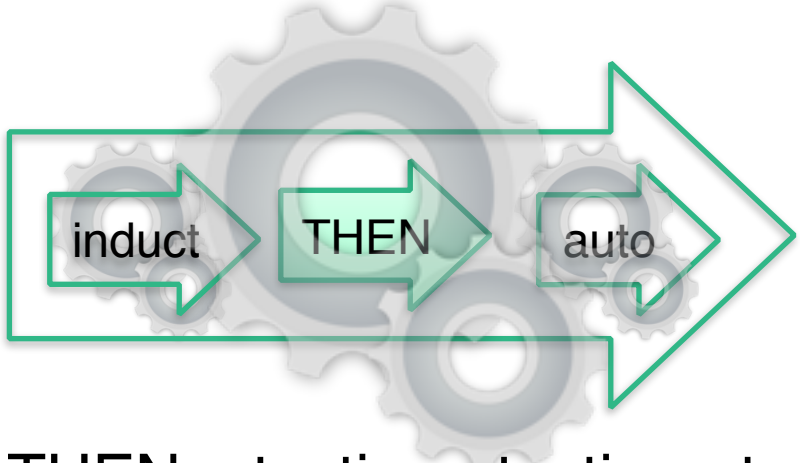
You can add new tactics

Tactical

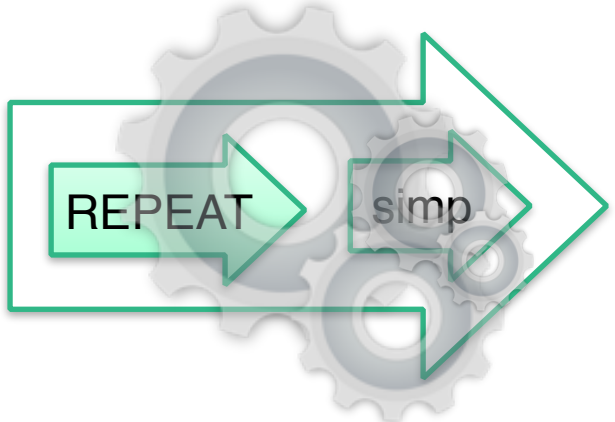
type tactic = thm -> [thm]



fun OR :: tactic -> tactic -> tactic



fun THEN :: tactic -> tactic -> tactic

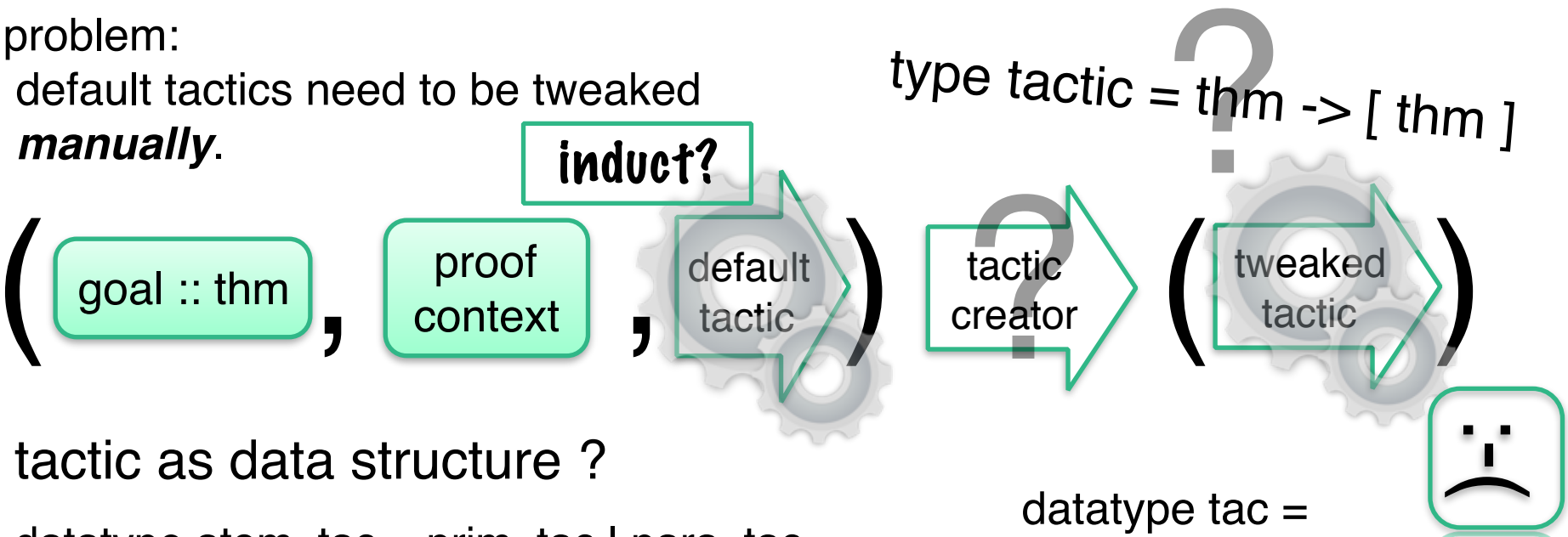


fun REPEAT :: tactic -> tactic

generic tactic?

Tactics 5

problem:
default tactics need to be tweaked ***manually***.



tactic as data structure ?

```
datatype atom_tac = prim_tac | para_tac

datatype prim_tac =
    Simp
  | Clarsimp
  | Fastforce
  | Induct

datatype para_tac =
    Para_Simp
  | Para_Clarsimp
  | Para_Fastforce
  | Para_Induct
```

```
datatype tac =
    Atom atom_tac
  | Succeed
  | Fail
  | Then (tac * tac)
  | Or   (tac * tac)
  | Rep  tac;
```

Monadic interpretation 1

type tactic = thm -> [thm]

tactic / tactical	type	monad operator
succeed	tactic	\ goal -> return goal
THEN	tactic -> tactic -> tactic	>=>
fail	tactic	\ goal -> mzero
OR ?	tactic -> tactic -> tactic	mplus ?
APPEND	tactic -> tactic -> tactic	mplus

((tactic1 OR tactic2) THEN tactic3) goal = ?

datatype ‘a tactic = (‘a -> ‘a monad0plus)

Monadic interpretation 2

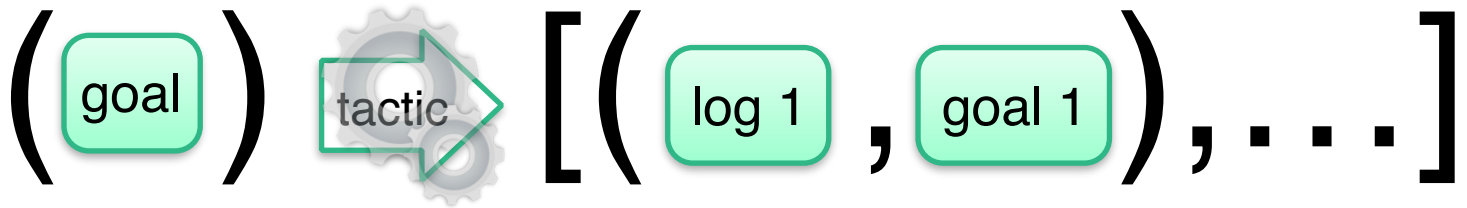
```
datatype tac =
    Atom atom_tac
  | Succeed
  | Fail
  | Seq (tac * tac)
  | Or   (tac * tac)
  | Rep  tac;
```

```
fun inter :: tac -> ‘a -> ‘a monad0plus
fun inter (Atom atom)     goal = eval atom goal
  | inter Succeed         goal = return goal
  | inter Fail             _   = mzero
  | inter (tac1 Seq tac2) goal = bind (inter tac1 goal) (inter tac2)
  | inter (tac1 Or  tac2) goal = mplus (inter tac1 goal, inter tac2 goal)
  | inter (Rep tac) goal       = inter ((tac Seq (Rep tac)) Or Succeed) goal
```

Slow !

Tactics 6

problems:
1. Poor feedback
2. Slow proof-check



type ‘a writerList = (‘a List) writerT

type tactic = thm -> thm writerList

Second Try:

Demo 2

Try the “try_hard” method

Future work

- In the *near* future …
  - more parameterised atomic methods
  - counterexample finder and ATPs
  - configuration flag for multiple proof-obligations
  - pretty printing of apply-script
  - Eisbach
  - evaluation
  - static analysis

- In the *distant* future …
  - lemma-suggestion
  - try hard -> try smart
    - quantifier
    - assertion tactic
    - proof-plan
    - timeout
    - how to parametrise methods

Conclusions again

- I am developing a proof automation tool for Isabelle/HOL.
- I am using monads for this.
- It can discharge proof obligation that Isabelle's default automation tools cannot prove.

Selected References

- Wadler, P. How to Replace Failure by a List of Successes
- Martin, A., and Gibbons, J. A monadic interpretation of tactics.
- Martin, A. P., Gardiner, P. H. B., and Woodcock, J. A tactic calculus
- Dreyer, D., Harper, R., Chakravarty, M. M. T., and Keller, G. Modular type classes.
- Kiselyov, O. et al Backtracking, Interleaving, and Terminating Monad Transformers

Thank You

SSRG/ProofEngineering
Yutaka Nagashima
Research Assistant

email firstname.surname@nicta.com.au

www.csiro.au