Representing Proof-Planning in the ρ-Calculus

Benjamin Wack, Serge Autexier, Andreas Meier

serge@ags.uni-sb.de

DFKI GmbH & Saarland University, Saarbrücken, Germany

2nd ρ-Calculus Workshop

LIX-Ecole Polytechnique, Palaiseau, France

What is Proof Planning?

- Theorem proving process that
  - conceives proofs at an abstract mathematics-oriented level
  - makes use of mathematical knowledge
  - uses AI planning technologies for proof search
  - postpones verification of abstract proof steps until an abstract proof has been found

$\Rightarrow$ Differs considerably from traditional calculus-level proof search in resolution/tableaux/matrix provers

- Different approaches embodied in the systems CLAM, PPLANNER, λ-CLAM, ISAPLANNER, MULTI

Motivation

- We want to model proof planning in a uniform framework such that we can
  - compare/relate the proof planning approach with other proof search strategies
  - compare different proof planning approaches among each other
- Modeling in the ρ-Calculus (already used to model ELAN strategies)

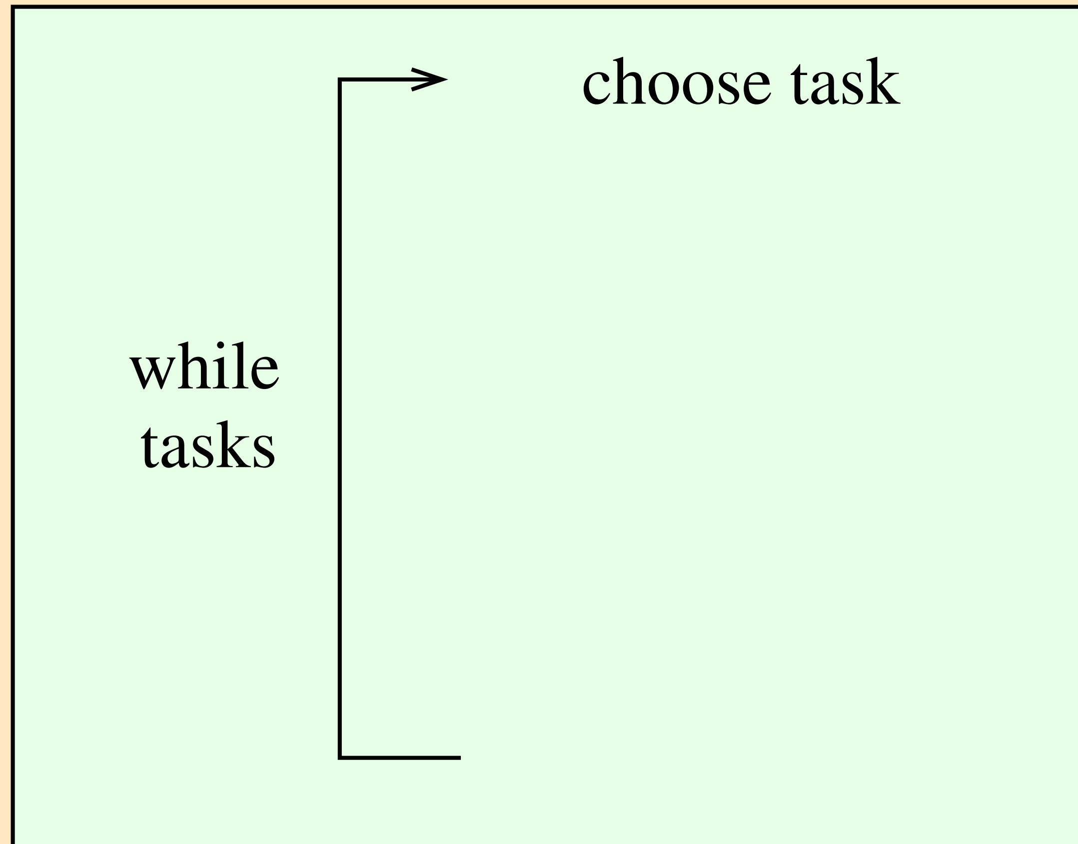
Proof Planning in ΩMEGA

- Recent planner: MULTI [A.Meier 2004]
- Uses mathematical proof knowledge in
  - methods to encode abstract proof steps
  - control rules declaratively encode heuristic control knowledge
  - strategies encode proof plan refinement services
- Major domains considered:
  ϵ–δ–proofs, residue class problems, permutation group problems, homomorphism theorems, Irrationality of $\sqrt[j]{k}$

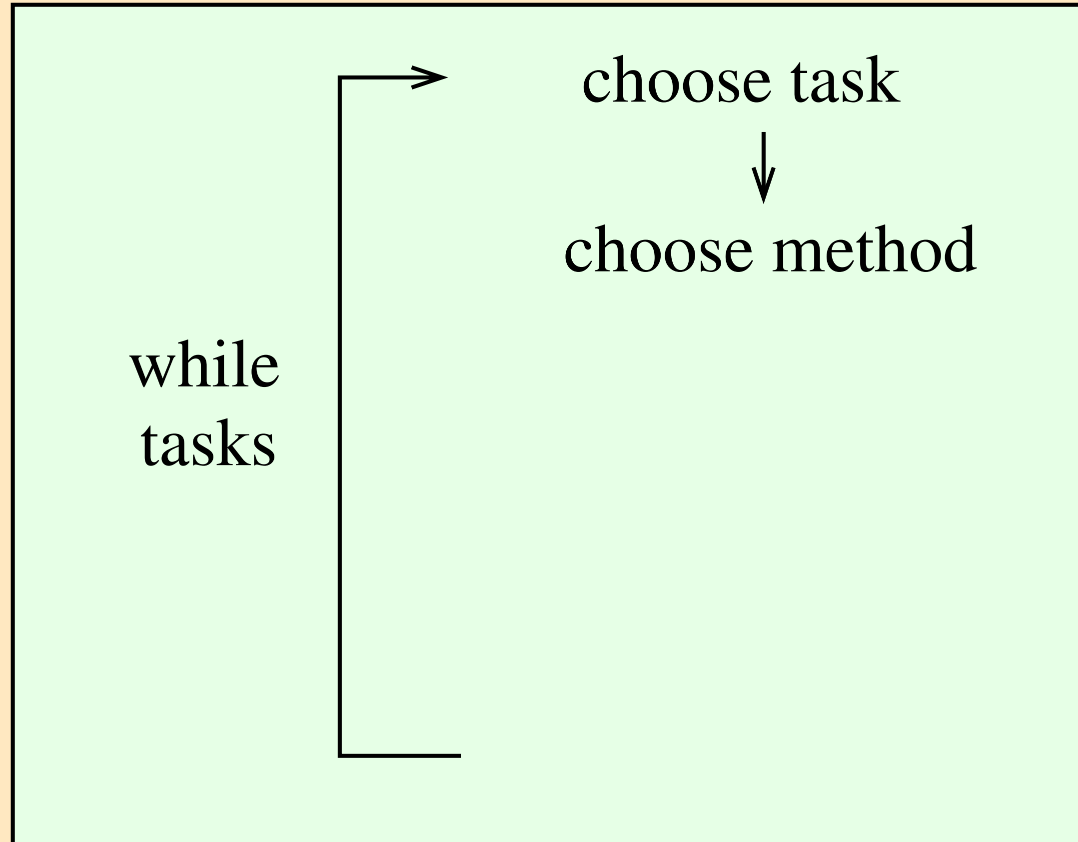
Proof Planning in ΩMEGA

- Henceforth:
  restrict to a simple proof planning algorithm at the level of methods only

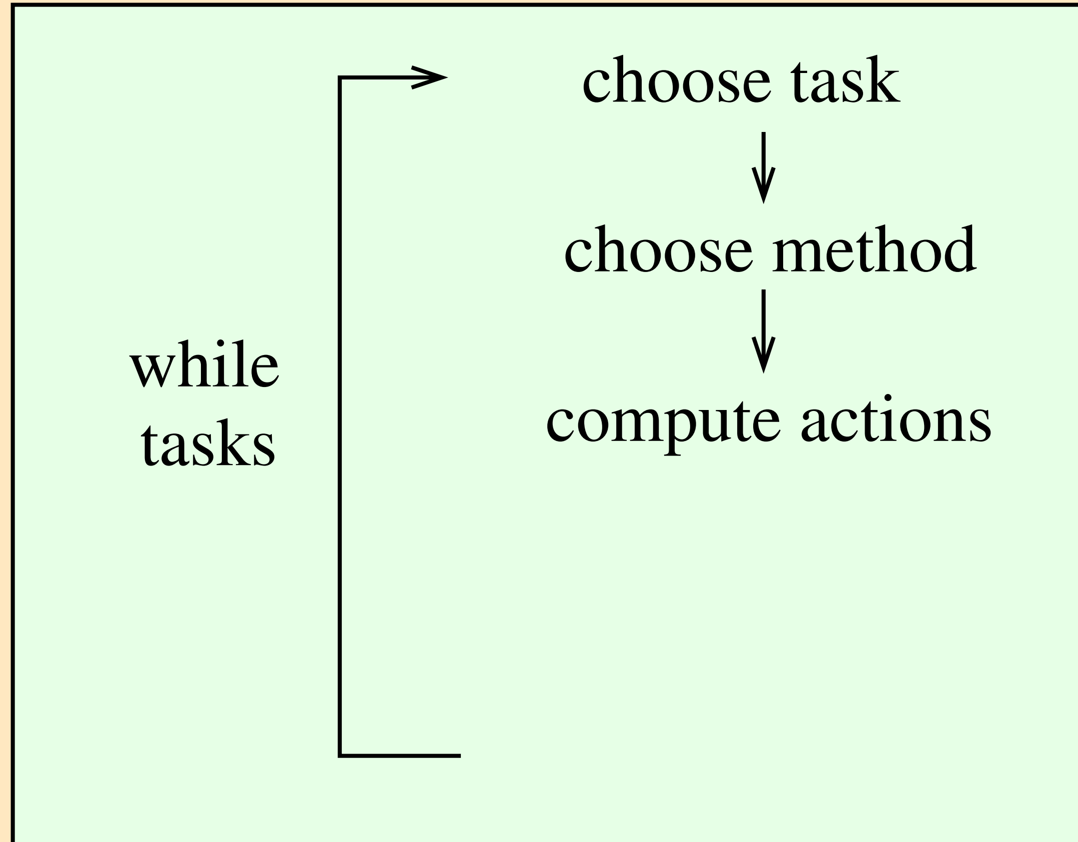
Simple Proof Planner



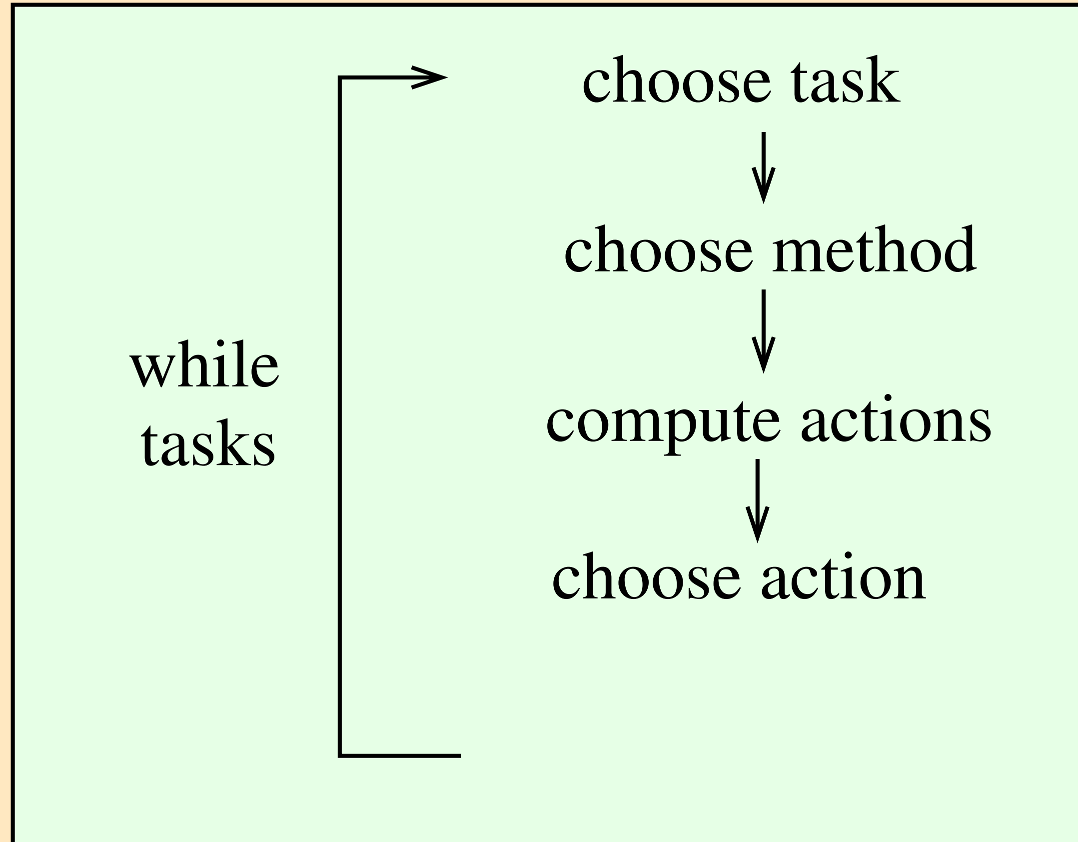
Simple Proof Planner



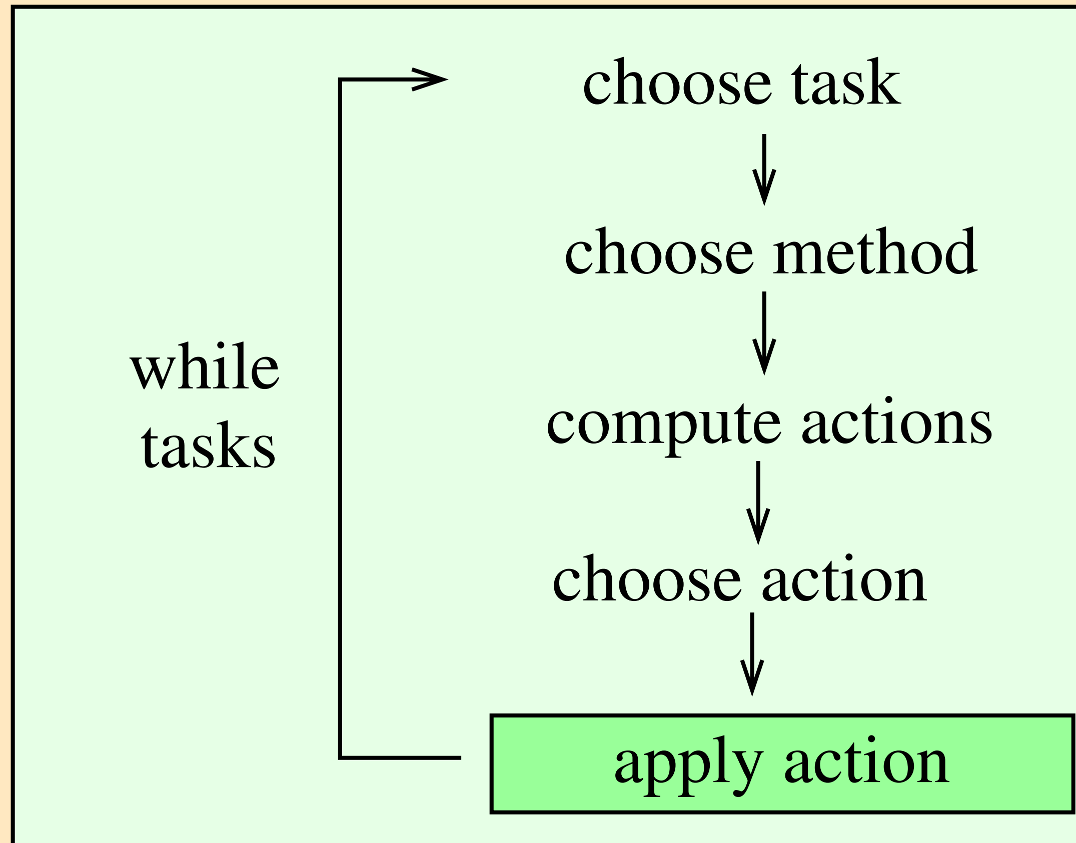
Simple Proof Planner



Simple Proof Planner



Simple Proof Planner



Simple Proof Planner

while tasks

choose task

choose method

compute actions

choose action

apply action

Backtrack Step

when no action

Simple Proof Planner

choose task
choose method
compute actions
choose action
apply action
while tasks
Backtrack Step
when no action

Guidance by control rules

Abstract simple proof planner

While *get-tasklist*(A) $\neq \emptyset$ do

1. Choose task T from A using the control rules CL;
2. Choose method M from ML using the control rules CL;
3. Apply method M to T which results in a list of actions, i.e., instantiated methods, denoted by $\mathsf{M}(\overrightarrow{p_1}), \ldots \mathsf{M}(\overrightarrow{p_n})$;
4. Choose an action $\mathsf{M}(\overrightarrow{p_i})$ using control rules from CL and apply it to T

   to obtain a list of new tasks TL;
5. $\mathsf{A} := \mathsf{A} - \{\mathsf{T}\} \cup \mathsf{TL}$.

Tasks

- Task = Single conclusion sequent with a label (proof line) and a justification

$$\mathsf{Lbl} : \langle \mathsf{G} \lhd \mathsf{P}_1, \ldots, \mathsf{P}_n \rangle (\mathsf{Just})$$

- Premises: $\mathsf{P}_1, \ldots, \mathsf{P}_n$
- Goal: G
- Label: Lbl
- Justification Just: the name of a method and a list of labels

$$\mathsf{Open}, \forall\mathsf{I}, \mathsf{ComplexEstimate}, \mathsf{Otter}, \mathsf{Maple}, \ldots$$

Agenda

- Simple approach: The agenda is a set of tasks
- Operations on the agenda:
  - Selection of a task

$$(X, P, X') \to P$$

  selects all the members of the list that match the pattern P. If no member matches P, the result is $\perp$.

  - removal of a task with label ℓ is achieved by

$$X, \ell : T(J), X' \to X, X'$$

  where T and J are variables.

Computations and External Systems

- Methods often uses external procedures, for computing a witness or a side condition.
- Example: a unification procedure *Unify*$(\cdot,\cdot)$, Call to Automated Theorem Provers, or Computer Algebra System.
- Calls to a procedure will be denoted $\mathsf{Proc_name}(\mathsf{arg}_1, \ldots, \mathsf{arg}_n)$.
- Until the procedure is defined (or if it is a call to an external system), the name $\mathsf{Proc_name}$ is considered as a constant with some matching theory defining its behaviour.
- For instance *Unify* is defined by the axioms:

$$
\begin{array}{lll}
\mathit{Unify}(\mathsf{t}, \mathsf{t}') & = \sigma & \text{if } \sigma\mathsf{t} = \sigma\mathsf{t}' \text{ and } \forall\theta \ (\theta\mathsf{t} = \theta\mathsf{t}' \Rightarrow \exists\theta',\ \theta = \theta' \circ \sigma) \\
 & = \bot & \text{otherwise}
\end{array}
$$

Encoding Methods

- A method takes a task, checks if it fulfills some conditions (some of which can be described by pattern matching), makes some computations and returns a list of new tasks.
- Thus its general shape is:

$$\langle G \lhd P \rangle \rightarrow (X \rightarrow TL)\ Comp(G, P)$$

- Example: A method for sidewards application of $<$-transitivity $\dfrac{a < c \quad c < b}{a < b}$

$$\langle a < b \lhd X, a' < c, X' \rangle \rightarrow \mathit{first} \begin{pmatrix} \bot \rightarrow \mathit{fail}; \\ \sigma \rightarrow \langle \mathit{Apply}(\sigma, c) < \mathit{Apply}(\sigma, b) \lhd X, \mathit{Apply}(\sigma, a') < \mathit{Apply}(\sigma, c), X' \rangle \end{pmatrix} \mathit{Unify}(a, a')$$

Encoding control rules

- Control rules modify a list of tasks, methods or actions in order to rank them by priority (and possibly add or remove some from the list).
- Their application is guided by conditions on the current agenda plan.
- Their general shape is:

$$\mathit{CR}(\ \mathsf{Kind}, \mathsf{A} \rightarrow \mathsf{L} \rightarrow ((\mathsf{True} \rightarrow \mathsf{Reorder}(\mathsf{L}); \mathsf{False} \rightarrow \mathsf{L})\mathsf{Cond}(\mathsf{A})))$$

  where $\mathsf{Kind} \in \{\mathsf{Task}, \mathsf{Meth}, \mathsf{Action}\}$.

- Cond is a (TRS computing a) boolean predicate on an agenda.
- $\mathsf{Reorder}$ is a modification to be applied on the list. . .

Rordering Lists

Reorder is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \to P$

where P and Q are patterns describing which group of Kind are manipulated

Rordering Lists

Reorder is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \to P$
- *Rejection* of a subset, performed by repeated application of $X, P, X' \to X, X'$

where P and Q are patterns describing which group of Kind are manipulated

Rordering Lists

Reorder is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \rightarrow P$
- *Rejection* of a subset, performed by repeated application of $X, P, X' \rightarrow X, X'$
- *Preference* of a subset, performed by repeated application of $X, P, X' \rightarrow P, X, X'$

where P and Q are patterns describing which group of Kind are manipulated

Rordering Lists

$\mathsf{Reorder}$ is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \to P$
- *Rejection* of a subset, performed by repeated application of $X, P, X' \to X, X'$
- *Preference* of a subset, performed by repeated application of $X, P, X' \to P, X, X'$
- *Deference* of a subset, performed by repeated application of $X, P, X' \to X, X', P$

where P and Q are patterns describing which group of Kind are manipulated

Rordering Lists

Reorder is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \to P$
- *Rejection* of a subset, performed by repeated application of $X, P, X' \to X, X'$
- *Preference* of a subset, performed by repeated application of $X, P, X' \to P, X, X'$
- *Deference* of a subset, performed by repeated application of $X, P, X' \to X, X', P$
- *Reordering* of a subset in front of another, performed by repeated application of $X, P, X', Q, X'' \to X, Q, P, X', X''$

where P and Q are patterns describing which group of Kind are manipulated

Rordering Lists

Reorder is a modification to be applied on the list from

- *Selection* of a subset, performed by $X, P, X' \rightarrow P$
- *Rejection* of a subset, performed by repeated application of $X, P, X' \rightarrow X, X'$
- *Preference* of a subset, performed by repeated application of $X, P, X' \rightarrow P, X, X'$
- *Deference* of a subset, performed by repeated application of $X, P, X' \rightarrow X, X', P$
- *Reordering* of a subset in front of another, performed by repeated application of $X, P, X', Q, X'' \rightarrow X, Q, P, X', X''$
- *Insertion* of a new element e, performed by $X \rightarrow e, X$

where P and Q are patterns describing which group of Kind are manipulated

Example Control Rule

A control rule promoting $\forall$I for agendas fulfilling condition $\mathtt{C}$

$$
CR(\ \mathsf{Meth},\ \mathsf{A} \rightarrow \mathsf{ML} \rightarrow \begin{pmatrix} \mathsf{True} \rightarrow (\mathsf{X}, \forall\mathsf{I}, \mathsf{X}' \rightarrow \forall\mathsf{I}, \mathsf{X}, \mathsf{X}')\mathsf{ML}; \\ \mathsf{False} \rightarrow \mathsf{ML} \end{pmatrix} \mathtt{C}(\mathsf{A})\ )
$$

Encoding the Simple Proof Planner

$\mathsf{A} \rightarrow CL \rightarrow \mathsf{ML} \rightarrow$

$(\quad \mathsf{T} \rightarrow$

$\quad (\quad \mathsf{M} \rightarrow$

$\quad\quad (\quad \textit{TL-Set} \rightarrow$

$\quad\quad\quad (\quad TL \rightarrow \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL)) \quad)\textit{select-tasklist}(\textit{TL-Set}, CL)$

$\quad\quad)\textit{apply-method}(\mathsf{M}, \mathsf{T})$

$\quad)\textit{select-method}(ML, \mathsf{T}, CL)$

$)\textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)$

Encoding the Simple Proof Planner

$$
\begin{array}{l}
\mathsf{A} \rightarrow CL \rightarrow \mathsf{ML} \rightarrow \\
(\;\; \mathsf{T} \rightarrow \\
\quad (\;\; \mathsf{M} \rightarrow \\
\quad\quad (\;\; \textit{TL-Set} \rightarrow \\
\quad\quad\quad (\;\; TL \rightarrow \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL)) \;\;)\textit{select-tasklist}(\textit{TL-Set}, CL) \\
\quad\quad)\textit{apply-method}(\mathsf{M}, \mathsf{T}) \\
\quad)\textit{select-method}(ML, \mathsf{T}, CL) \\
)\textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)
\end{array}
$$

where the following procedures are used:

get-tasklist is simply the identity in this case, since the agenda is already a list of tasks.

Encoding the Simple Proof Planner

$$
\begin{array}{l}
\mathsf{A} \rightarrow CL \rightarrow \mathsf{ML} \rightarrow \\
(\ \ \mathsf{T} \rightarrow \\
\quad (\ \ \mathsf{M} \rightarrow \\
\quad\quad (\ \ \textit{TL-Set} \rightarrow \\
\quad\quad\quad (\ \ TL \rightarrow \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL))\ \ )\textit{select-tasklist}(\textit{TL-Set}, CL) \\
\quad\quad)\textit{apply-method}(\mathsf{M}, \mathsf{T}) \\
\quad)\textit{select-method}(ML, \mathsf{T}, CL) \\
)\textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)
\end{array}
$$

where the following procedures are used:

select-task applies successively all the task control rules over the task list and finally takes the first:

$$
\mathsf{TL} \rightarrow \mathsf{first}\left(\begin{array}{l}
\mathsf{nil} \rightarrow \mathsf{TL}; \\
\textit{CR}(\ \ \mathsf{Task}, \mathsf{R})\ \ , \mathsf{CL}' \rightarrow \textit{select-task}((\mathsf{R}\ \mathsf{TL}), \mathsf{CL}'); \\
\mathsf{C}, \mathsf{CL}' \rightarrow \textit{select-task}(\mathsf{TL}, \mathsf{CL}')
\end{array}\right)
$$

Encoding the Simple Proof Planner

$$
\begin{array}{l}
\mathsf{A} \rightarrow CL \rightarrow \mathsf{ML} \rightarrow \\
(\ \ \mathsf{T} \rightarrow \\
\quad (\ \ \mathsf{M} \rightarrow \\
\quad\quad (\ \ \textit{TL-Set} \rightarrow \\
\quad\quad\quad (\ \ TL \rightarrow \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL))\ \ )\textit{select-tasklist}(\textit{TL-Set}, CL) \\
\quad\quad)\textit{apply-method}(\mathsf{M}, \mathsf{T}) \\
\quad)\textit{select-method}(ML, \mathsf{T}, CL) \\
)\textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)
\end{array}
$$

where the following procedures are used:

select-method applies successively all the method control rules over the method list and then tries every method on the current task. Keeps first successful method

$$
\mathsf{ML} \rightarrow \mathsf{T} \rightarrow \mathsf{first}\left(\begin{array}{l}
\mathsf{nil} \rightarrow \mathsf{first}(\mathsf{ML})\mathsf{T}; \\
\textit{CR}(\ \ \mathsf{Meth}, \mathsf{R})\ \ , \mathsf{CL}' \rightarrow \textit{select-method}((\mathsf{R}\ \mathsf{ML}), \mathsf{T}, \mathsf{CL}'); \\
\mathsf{C}, \mathsf{CL}' \rightarrow \textit{select-method}(\mathsf{ML}, \mathsf{T}, \mathsf{CL}')
\end{array}\right)
$$

Encoding the Simple Proof Planner

$$
\begin{array}{l}
\mathsf{A} \to CL \to \mathsf{ML} \to \\
(\ \mathsf{T} \to \\
\quad (\ \mathsf{M} \to \\
\quad\quad (\ \textit{TL-Set} \to \\
\quad\quad\quad (\ TL \to \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL))\ )\textit{select-tasklist}(\textit{TL-Set}, CL) \\
\quad\quad)\textit{apply-method}(\mathsf{M}, \mathsf{T}) \\
\quad)\textit{select-method}(ML, \mathsf{T}, CL) \\
)\textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)
\end{array}
$$

where the following procedures are used:

select-tasklist applies successively all the action control rules over the list of task lists and finally takes the first:

$$
\textit{TL-Set} \to \mathsf{first} \left(\begin{array}{l}
\mathsf{nil} \to \textit{TL-Set}; \\
\textit{CR}(\ \mathsf{Action}, \mathsf{R})\ , \mathsf{CL}' \to \textit{select-tasklist}((\mathsf{R}\ \textit{TL-Set}), \mathsf{CL}'); \\
\mathsf{C}, \mathsf{CL}' \to \textit{select-tasklist}(\textit{TL-Set}, \mathsf{CL}')
\end{array} \right)
$$

Encoding the Simple Proof Planner

$\mathsf{A} \rightarrow CL \rightarrow \mathsf{ML} \rightarrow$
$(\quad \mathsf{T} \rightarrow$
$\quad (\quad \mathsf{M} \rightarrow$
$\quad\quad (\quad \textit{TL-Set} \rightarrow$
$\quad\quad\quad (\quad TL \rightarrow \textit{justify}(\mathsf{T}, \mathsf{A}, \textit{justification}(\mathsf{T}, \mathsf{M}, TL)) \quad) \textit{select-tasklist}(\textit{TL-Set}, CL)$
$\quad\quad) \textit{apply-method}(\mathsf{M}, \mathsf{T})$
$\quad) \textit{select-method}(ML, \mathsf{T}, CL)$
$) \textit{select-task}(\textit{get-tasklist}(\mathsf{A}), CL)$

where the following procedures are used:

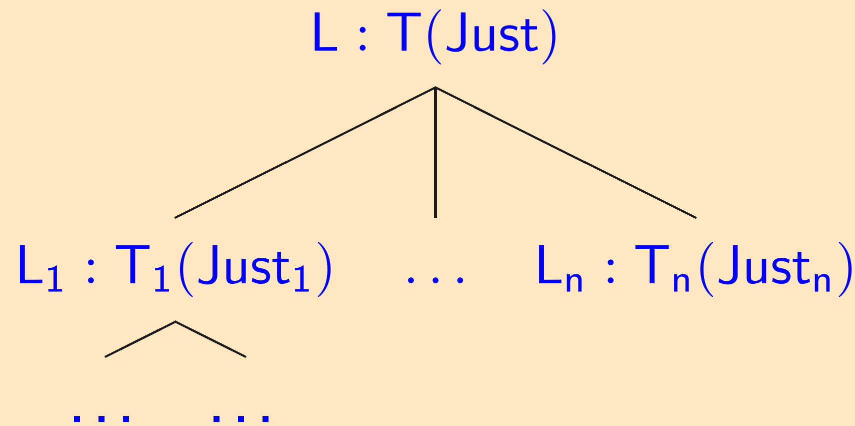
justify replaces an open task with the selected justification and adds the list of new open tasks to the current agenda.

A more complex agenda

- How to backtrack can also be defined by control rules (defining a *backtracking strategy*)
- Independently *where* the backtracking strategies decide to prune the search tree, they all do dependency-based pruning of subtrees
- This is not supported by using an agenda composed only of the current list of tasks
- Need to track dependencies by using a tree data structure which is closer to the proof datastructure (PDS)
- A node of the tree is still a labelled task with a justification, but . . .

Agenda as Dependency Tree

- A node of the tree is still a labelled task with a justification, but the justification can contain new nodes (instead of labels)



is represented as

$$L : T(Just(L_1 : T_1(Just_1(\ldots, \ldots)), \ldots, L_n : T_n(Just_n)))$$

- The search of a given node now requires a traversal function

Examples

- Example: Selection of all the nodes that match the pattern P.

$$(\mathsf{S} \to \left(\begin{array}{l} \mathsf{P} \to \mathsf{P}; \\ \mathsf{L} : \mathsf{T}(\mathsf{J}(\mathsf{X}, \mathsf{N}, \mathsf{X}')) \to \mathsf{S}\ \mathsf{S}\ \mathsf{N} \end{array} \right)) \ (\mathsf{S} \to \left(\begin{array}{l} \mathsf{P} \to \mathsf{P}; \\ \mathsf{L} : \mathsf{T}(\mathsf{J}(\mathsf{X}, \mathsf{N}, \mathsf{X}')) \to \mathsf{S}\ \mathsf{S}\ \mathsf{N} \end{array} \right))$$

- Pruning the subtree at a given node (once this node is found) is achieved by the simple term $\mathsf{L} : \mathsf{T}(\mathsf{J}) \to \mathsf{L} : \mathsf{T}(\mathsf{Open})$.

Comparison with Elan

PPlanning	Rho	Elan
Agenda	List term	
Agenda	Tree term	Algebraic term
Precondition	Pattern	Conditional rewriting
Method	Abstraction	Rule
Main loop	Fixed points	`repeat*`
Control rules	List management	
	Structures + $\mapsto_\delta$	`dk` / `dc`
Dependency-based pruning	Pruning	
Try + Backtracking Control Rules (implicit first)	first / stk	`try` / `first`

Future

Modelling Proof Planning Procedures

- Include expansion of proof planning steps
- Treatment of Metavariables
- MULTI: Describe the level of *Strategies*
  - Strategy = Set of methods + Control rules
  - MULTI selects strategy for the simple proof planner
- λ-CLAM, ISAPLANNER: Model continuations explicitly
  - For ISAPLANNER: Support
    - access to continuation by simple planner
    - manipulation of continuation by the planner

Compare with other search procedures

- How well can (specific) search procedures, control, backtracking be encoded in the different formalisms?

Conclusion

- So far modelling in the ρ-Calculus is (mostly) straightforward
- Allows for a very concise representation of the main aspects
- Even for the simple planner we have already initial results by comparing it with other search strategies
  Language constructs provided/used for modelling search
- We are at the beginning but it looks promising that we can get nice results