*Electronic Notes in Theoretical Computer Science 58 No. 2 (2001)*

*URL:* [*http://www.elsevier.nl/locate/entcs/volume58.html*](http://www.elsevier.nl/locate/entcs/volume58.html) *14 pages*

*A Proof-Planning Framework with explicit* Abstractions based on Indexed Formulas

# *Serge Autexier* [*1*](#_bookmark0)

*FR 6.2 Informatik, Saarland University*

*P.O. Box 15 11 50, 66041 Saarbrucken, Germany*

*Abstract*

*A major motivation of proof-planning is to bridge the gap between high-level, co- gnitively adequate reasoning for speci c domains, and calculus-level reasoning to ensure soundness. For high reasoning levels the cognitive adequacy of representa- tion and reasoning techniques is a major issue, while for lower reasoning levels the adequacy wrt. the modelled domain is important. Furthermore, proof construction is an engineering task and there is a need to support the design and application of proof-search engineering methods. To this end we present a framework to expli- citly support di erent reasoning levels. To structure reasoning levels the framework allows for an explicit representation of abstractions and proof-search re nement techniques. In order to ensure soundness within a reasoning level, we use techni- ques developed in the context of matrix characterisation relying on the notion of indexed formulas. Furthermore, we introduce a uniform concept for contextual rea- soning, and sketch basic tacticals for the de nition of tactics to organise the overall proof-search inside and across di erent reasoning levels.*

*Keywords: Proof-planning, Methodology, Abstraction, Matrix Characterisations*

# *1 Introduction*

*A major motivation for proof-planning is to allow on the one hand for a* high-level reasoning which is (cognitively) adequate for speci c problems and domains. On the other hand the nal product of the reasoning process must be a calculus level proof wrt. some logic, in order to have a veri ably sound proof. Besides the cognitive motivation, another reason for the di erent re- asoning levels is that the construction of a proof for some theorem is an en- gineering task. Analogously to the development of software, where one starts with a high-level speci cation of a program that is subsequently re ned to

*1 Email:* [*Autexier@ags.uni-sb.de*](mailto:Autexier@ags.uni-sb.de)

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license*](http://creativecommons.org/licenses/by-nc-nd/3.0/)*.*

*an executable program in some target programming language, we view proof-* planning as a methodology to design (and subsequently apply) di erent re- asoning levels as well as methods to stepwise re ne proof speci cations (i.e. proof plans). Hence, proof-planning has been introduced as a methodology to re ne high-level proof-plans of a theorem to a calculus level.

*The objectives of the di erent reasoning levels are di erent: for high rea-* soning levels, the adequacy of the representation and the reasoning techniques is a key issue, in order to allow for an intuitive reasoning, which is as close as possible to say the style of a human. Although the reasoning inside one reasoning level should be sound, there is another notion of relative soundness due to the presence of di erent reasoning levels. A high reasoning level is relatively sound wrt. some lower reasoning level, if any high level proof can be re ned to some low level proof. Usually, this relative soundness does not hold, which is a well-known e ect in the presence of abstractions [[5].](#_bookmark15) We do not want to enforce the relative soundness of high reasoning levels, as this usually may hamper the adequacy of representations, whereas the primary purpose of those levels is to allow for an eÆcient and adequate search for a proof plan. The relative soundness of a proof on a high reasoning level is established by stepwise re nement of this proof to a calculus level proof. On the lower reasoning levels we are less concerned with the cognitive adequacy of representations, but rather with the adequacy wrt. the modelled domain, as for example some program behaviour or a mathematical domain.

*In this paper we present a proof-planning framework that supports the* design and application of proof search engineering methods. First, we de ne in Sect. [2](#_bookmark1) the notion of a reasoning level, which consists of reasoning objects (e.g formulas, sequents, etc.) and reasoning procedures (e.g. tactics, plan- ning algorithms, general theorem proving algorithms, etc.). For example, in a proof-planner like [[10,2],](#_bookmark12) the reasoning objects are sequents as well as the pre- and postconditions of some methods, which are used as inference rules. The reasoning procedure of a proof-planner is the depth- rst search algorithm applying the methods. To ensure the soundness wrt. some logic within a reaso- ning level, a variety of logics are prede ned in the framework, and a uniform mechanism to encode those logics is introduced. For sake of representatio- nal adequacy, the logics are not embedded in some meta-logic. Instead we present in Sect. [4](#_bookmark4) techniques developed in the context of matrix characteri- sation [[12](#_bookmark22)[,9]](#_bookmark19) that rely on the notion of indexed formulas. Those allow for a uniform treatment of a large class of classical, modal and intuitionistic logics, without having to change the syntax of the formulas.

*Secondly, we de ne in Sect.* [*3*](#_bookmark2) *vertical structuring mechanisms to relate* di erent reasoning levels. This includes on the one hand the explicit de nition of abstractions, which de ne how reasoning objects of a higher reasoning level are related to those of a lower level. On the other hand, we introduce the notion of re nements, which determine how higher-level reasoning procedure can be implemented by some lower-level reasoning procedures. This subsumes

*e.g. the notion of re nement encoded in a method of a proof-planner like [*[*10*](#_bookmark20)[*,2],*](#_bookmark12) where the tactic wrapped inside a method determines the re nement of the abstract rule described by the pre- and postcondition of the method. Based on the general notion of re nement, the re nement of an abstract proof can be de ned in a uniform manner.

*Thirdly, reasoning procedures should be de nable in an intuitive manner,* again because of the overall motivation of cognitive adequacy. To this end we present in Sect. [5](#_bookmark7) window inference reasoning, which is a formalisation of an intuitive reasoning style, that allows to focus the reasoning on sub-formulas and sub-terms, and aims at contextual reasoning. We integrate window in- ference reasoning with the indexed formulas from Sect. [4](#_bookmark4) and present as a by-product how this allows for contextual reasoning in a canonical way.

*Finally, we present in Sect.* [*6*](#_bookmark10) *the basic programming language to de ne* reasoning and re nement procedures and how to combine those in order to organise the overall proof search. These build upon the intuitive reasoning rules provided by window inference reasoning and have an explicit failure and success semantics.

# *2 Reasoning Levels*

*In this section we introduce the notion of reasoning levels. Those are the horizontal structures of the framework in opposition to the abstraction and re nement techniques (cf. Sect.* [*3*](#_bookmark2)*) that are the vertical structures.*

*A reasoning level consists on the one hand of the objects we want to reason* about as well as the basic reasoning rules, and on the other hand of the reasoning procedures organising the proof search within one reasoning level. The former are de ned by a signature and a set of formulas wrt. some logic. Thus, the reasoning objects and rules of a reasoning level are given by a pair ( ; ), where is the signature and is a set of \axioms", and we denote such a pair as a representation. This is similar to the notion of a speci cation in formal speci cation languages [[8],](#_bookmark18) which are e.g. used to de ne abstract datatypes and functions about abstract datatypes. For a reasoning level, the signature determines the syntax of the reasoning objects and the available basic reasoning rules are determined by the \axioms" from .

*Example 2.1 As an example we present a representation called TLA for TLA* formulas [[7]](#_bookmark17) and a representation TLA-States&Transitions for the abstrac- tion of TLA formulas. The signature for TLA contains the usual logical connec- tives, as well as 2; 3, and also WF to express fairness requirements. As an example we use a simple program, which has a counter x of initial value *15*

*and that decreases this counter in every step if it is greater than 0. This is*

*encoded in TLA by the following formula*

*x = 15 ^ 2(x 0 ^ x0 = x 1) ^ WF(x 0 ^ x0 = x 1)x*

*In order to prove that the counter eventually has the value 0 (encoded by*

*3x = 0), we use as abstraction a representation consisting of states and* state transitions. This representation is called TLA-States&Transitions and consists of states hni representing the state, where x has the value n, and state transitions represented by 8n : hni ) hn 1i. This is an abstraction of the transition made by the program, i.e. x 0 ^ x0 = x 1. Finding a proof-plan for 3x = 0, consists in nding a proof for h0i on the (abstract) representation TLA-States&Transitions. *2*

*The second component of a reasoning level are the reasoning procedures,* which organise the proof search within the reasoning level. We adopt a ge- neral notion for a reasoning procedure, by de ning it to be an algorithm in some programming language, which interacts with the proof by applying ba- sic inference rules. Additionally a reasoning procedure comes with a notion of success and failure to re ect about its own behaviour. E.g. in case of failure a reasoning procedure might invoke backtracking to some previous proof state. This notion of a reasoning procedure is general enough to sub- sume various kinds of speci c reasoning procedures developed in automated theorem proving, like, e.g., a tactic in a tactical theorem prover, the various algorithms in an automated theorem prover, or a proof-planning algorithm in some proof-planner.

*In summary, we have de ned a reasoning level to consist of two parts,* namely representations de ning the syntax of the reasoning objects and rea- soning procedures to organise the proof-search inside the reasoning level. This can be sketched by the following equation:

*Reasoning level = Representation + Reasoning procedures*

*The next step consists in relating reasoning levels. To this end vertical* structures are introduced in the next section, that describe the mechanisms to switch between reasoning levels.

# *3 Structuring Reasoning Levels*

*In this section we de ne the mechanisms to structure the di erent reasoning* levels of the framework (cf. Fig. [1).](#_bookmark3) There are two dual notions of vertical structures: rst, abstractions can be de ned which describe how objects of a higher reasoning level can be constructed from objects of a lower reasoning levels. Formally, an abstraction is a mapping of a set of objects of a source representation RS to a set of objects of some target representation RT . For the purpose of the framework, we de ne an abstraction to consist of a name, the source and target representations RS and RT , and an abstraction procedure (i.e. an algorithm) to compute the mapping.

*Example 3.1 Consider as an example the source representation NAT of na-* tural numbers and labelled fragments (LF-NAT) as respective target represen-

*Reasoning Level 3 = Representation-3 + Reasoning Procedures* *3*

*Abstraction Re nement*

*Reasoning Level 2 = Representation-2 + Reasoning Procedures* *2*

*Abstraction Re nement*

*Reasoning Level 1 = Representation-1 + Reasoning Procedures* *1*

*Fig. 1. Structuring reasoning levels*

*tation. Labelled fragments are known abstractions in the context of inductive* theorem proving, where they are used to abstract over the speci c di erences (so-called wave-fronts) between two terms sharing a common skeleton. The representation NAT consists of a signature for natural numbers and { among others { of two axioms de ning the addition of natural numbers

*(1) 8y 0+ y = y*

*(2) 8x; y s(x)+ y = s(x + y)*

*The representation LF-NAT consists on the one hand of a signature extending* the signature of NAT by a function : Nat ! Nat that is used to represent the wave-fronts, and on the other hand of the axioms

*(3) 8y (y)= y*

*(4) 8x; y (x)+ y = (x + y)*

*(5) 8x (s(x)) = s( (x))*

*The labelled-fragment abstraction is an abstraction from NAT to LF-NAT and its associated abstraction procedure maps the singleton f*[*(1*](#_bookmark3)*)g to the singleton f*[*(3*](#_bookmark3)*)g, and the singleton f*[*(2*](#_bookmark3)*)g to the set f*[*(4*](#_bookmark3)*);* [*(5)*](#_bookmark3)*g.* *2*

*Thus, an abstraction consists of the source and target representations and* a function mapping objects from the source representation to objects of the target representation. This can be expressed in an intuitive manner by the following equation:

*Abstraction = Source & Target Representations + Abstraction procedure The dual vertical structuring mechanism is the re nement of higher-level*

*reasoning procedures to lower-level reasoning procedures. Take as an example*

*of a high-level reasoning procedure some planning algorithm, which computes* a proof-plan from methods. This proof-plan is a \proof" on the reasoning level of the proof-planning procedure, and consists of the application of basic rewriting rules de ned by the pre- and postconditions of the methods. We can associate to this planning procedure a re nement procedure, which takes generated proof-plan and re nes it by calling the tactics wrapped inside the di erent methods. In this scenario the re nement information is associated

*to each basic reasoning rule of the higher reasoning level via the methods.* A method describes that an abstract proof step satisfying a certain pattern (described by the pre- and postcondition of the method) could be re ned by the tactic wrapped inside the method. The re nement procedure exploits this knowledge in order to re ne a proof plan. In general it is not possible to describe this re nement information by patterns. In order to deal with the general case, we allow to associate a re nement procedure to some reasoning procedure. Such a re nement procedure takes the proof generated by the re- asoning procedure as a proof-plan to compute a proof on the lower reasoning level (using the lower-level reasoning procedures). Similarly to reasoning pro- cedures, re nement procedures come with an explicit success and failure semantics to re ect about their own behaviour. This allows for example to backtrack the re nement process and to try alternative re nements.

*>From the above we can summarise the notion of re nement by stating* that a re nement is composed of a reasoning procedure and an associated re nement procedure. This can be sketched by the following equation:

*Re nement = Reasoning procedure + Re nement procedure*

*Note that a re nement procedure for some reasoning procedure P can* dually be interpreted as some kind of abstraction, where all the (lower-level) reasoning procedures used by the re nement procedure are abstracted to the reasoning procedure P. This notion of abstraction is di erent from the notion presented above, since it is an abstraction of reasoning procedures while the former is an abstraction of objects and axioms. Furthermore do re nement procedures exploit knowledge about object abstractions during the re nement. Hence, an object abstraction also gives rise to some notion of re nement. The reason why we introduced abstractions mainly for objects and re nement mainly for reasoning procedures is because this is the usual way those are used.

# *4 Handling Soundness*

*In this section we de ne a uniform mechanism to handle soundness for a* variety of logics. For sake of the adequacy of representation, we want to support directly the soundness wrt. some logic without altering the syntax of formulas. Subsequently, we do not want to embed a logic in some meta-logic, like e.g. intuitionistic higher-order logic. In comparison to the representations introduced in Sect. [2](#_bookmark1), a logic also de nes the syntax of reasoning objects, but comes with its own formal proof-theory (whereas a representation is relative to some logic, and thus \inherits" the proof-theory from this logic). Thereby its proof-theory determines, for example, whether in a sequent calculus for this logic an elimination rule for some object-level quanti er must satisfy the Eigenvariable condition, or how other parts of a sequent are a ected by the elimination rule for some 2 or 3 in some modal logic.

|  |  |  |
| --- | --- | --- |
|  | *0* | *1* |
| *(' \_ )+* | *'+* | *+* |
| *(' ) )+* | *'* | *+* |
| *(' ^ )* | *'* |  |
| *(:')+* | *'* |  |
| *(:')* | *'+* |  |

|  |  |  |
| --- | --- | --- |
|  | *0* | *1* |
| *(' ^ )+* | *'+* | *+* |
| *(' \_ )* | *'* |  |
| *(' ) )* | *'+* |  |

|  |  |
| --- | --- |
|  | *0(c)* |
| *(8x ')* | *('[x=c]* |
| *(9x ')+* | *('[x=c]+* |

|  |  |
| --- | --- |
|  | *0* |
| *(2')* | *'* |
| *(3')+* | *'+* |

*Fig. 2. Uniform notation for signed formulas*

|  |  |
| --- | --- |
| *Æ* | *Æ0(c)* |
| *(8x ')+* | *('[x=c]+* |
| *(9x ')* | *('[x=c]* |

|  |  |
| --- | --- |
|  | *0* |
| *(2')+* | *'+* |
| *(3')* | *'* |

*A background mechanism to handle the proof-theory of di erent logics* must allow for a uniform handling of soundness and completeness results for those logics. Such a basis for propositional and rst-order, classical, modal and intuitionistic logics is provided by indexed formula trees [[12]](#_bookmark22) and relies on polarities of (sub-)formulas and uniform notation [[12](#_bookmark22)[,4,3].](#_bookmark13)

*The starting point for uniform notation is the concept of signed formulas,* which are formulas ' annotated by some positive ('+) or negative (' ) po- larity. Intuitively, a signed formula '+ occurs in the succedent of a sequent,

*i.e. ` '; , and ' occurs in the antecedent of a sequent, i.e. ;' ` .* The signed formulas can be categorised into di erent classes, according to their behaviour during the proof search: consider as an example the sequent

*` ' \_ . The application of the respective sequent decomposition rule leads* to the single sequent ` '; . All those kinds of signed formulas are said to be of type . The complete list of signed formulas of this type are given in the table for in Fig. [2](#_bookmark5).

*Furthermore, those signed formulas whose decomposition leads to a split* on the sequent proof tree are said to be of type (cf. Fig. [2).](#_bookmark5) Signed formulas of type correspond to instantiation rules, without restrictions to the substi- tuted terms, while for Æ-type signed formulas the instantiation must respect the Eigenvariable condition (e.g. (8x ')+, which corresponds to ` 8x '). For modal logic formulas, the decomposition of -type signed formulas does not a ect the other formulas in a sequent (e.g. ; 2' ` ), while the decom- position of -type signed formulas changes the other parts of the sequent (e.g.

*` 2'; ).*

*In Fig.* [*2*](#_bookmark5) *the rules also indicate how the polarity is inherited to the sub-* formulas of a signed formula. Using these rules, polarities can be assigned to all sub-formulas of a formula and the polarity of a subexpression indicates whether this sub-formula will occur in the antecedent or the succedent of a sequent, if all structural sequent decomposition rules are applied until this formula. E.g. the signed formula (' ) ) corresponds to ;' ) ` . The -rule reduces this signed formula into the signed formulas '+ and .

*a0*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *u* | *Pol(u)* | *Lab(u)* | | | | | | *Ptype(u)* | *Stype(u)* |
| *a0* | *+* | *(2(8x* | *P* | *(x))) ) (9y (P (y) ^* | *2P* | *(y* | *)))* |  |  |
| *1* |  | *2* | | | | | |  | *1* |
| *a2* |  | *8x P (x)* | | | | | |  | *0* |
| *a3* |  | *P (x)* | | | | | |  | *0* |
| *a0*  *3* | *Æ* | *x* | | | | | |  |  |
| *a4* | *+* | *9y (P (y) ^ 2P (y))* | | | | | |  | *2* |
| *a5* | *+* | *P (y) ^ 2P (y)* | | | | | |  | *0* |
| *a6* | *+* | *P (y)* | | | | | |  | *1* |
| *a0*  *6* | *+* | *y* | | | | | |  |  |
| *a7* | *+* | *2P (y)* | | | | | |  | *2* |
| *a8* | *+* | *P (y)* | | | | | |  | *0* |
| *a0*  *8* | *Æ* | *y* | | | | | |  |  |

*a1 a4*

*a2 a5*

*a3 a6 a7*

*0 0*

*a*

*a*

*a*

*3 6 8*

*0*

*a*

*8*

*Fig. 3. Indexed Formula tree*

*This corresponds to what happens if the )-Left-decomposition rule is applied* on ;' ) ` , since it results in a split of the proof (indicated by ), and ' occurs in the succedent of the rst premise ` '; , and occurs in the antecedent of the second premise ; ` .

*Indexed formula trees as introduced in* [*[12]*](#_bookmark22) *exploit the tree structure of* formulas and annotate each sub-formula according to the reduction rules with their respective polarity and uniform type. Consider the indexed formula tree in Fig. [3](#_bookmark6) of the signed formula ((28x P (x)) ) (9y (P (y) ^ 2P (y))))+. For each sub-formula there is a position ai that is either a position constant or position variable (which are over-lined, like a1). A set of properties associated to each position is listed in the adjacent table: those are the actual sub- formula (called the label of the position), its polarity, its uniform type (called primary type), and its secondary type, which is its type inside the parent formula reduction rule. According to those reduction rules, only sub-formulas that are related by an -type position may occur together in a nal sequent (after application of the respective sequent calculus decomposition rules). This can be illustrated by the sub-formulas 8x P (x) and P (y) that are related by the position a0, or, similarly, the sub-formulas 8x P (x) and 2P (y)). Hence, from the indexed formula tree we obtain a simple notion of a context for some subexpression, i.e. those parts of the signed formula tree, that can be

*\used" for this subexpression. This is exploited in Sect.* [*5*](#_bookmark7) *to handle contextual* reasoning.

*The tree structure of the indexed formula tree induces an ordering among* the positions. A rst-order substitution instantiates a variable bound on

*some position a of primary type and induces an additional ordering relati-* onship between the position a and - or Æ-type positions binding variables in

*(x). Combined with the original ordering , we obtain a relation . Such a*

*rst-order substitution is only sound, if is irre exive.*

*Furthermore, we can associate to any position a a string of positions, con-* sisting of some of the positions governing a (i.e. smaller wrt. ) and that are of primary type or . Such a string is called the modal pre x of the posi- tion a. Which positions are actually used in the pre x depends on the modal logic under consideration (K, K4, S4, etc., cf. [[12]](#_bookmark22) for details). The variable positions are exactly all positions of primary type . Modal substitutions are used to instantiate the position variables by modal position strings. Depen- ding on the modal logic under consideration, there is a set of restrictions how the position variables can be instantiated (cf. [[12]).](#_bookmark22) Furthermore, similar to

*rst-order substitutions a modal substitution induces an ordering among the*

*- and -type positions and again the induced overall ordering must be* irre exive. All the requirements about the instantiation of variables, position variables, the irre exivity of induced overall orderings among positions are subsumed by the uniform notion of L-admissible substitutions, where L is eit- her classical rst-order logic, or propositional or rst-order modal logic K, K4, D, D4, T, S4, or propositional or rst-order intuitionistic logic. The uniform notion of L-admissible substitutions is the key concept allowing for a uniform treatment of a large class of logics.

*Based on these techniques, we introduce the uniform notion of valid proof* states (i.e., a signed formula tree with an irre exive ordering ). From [[12](#_bookmark22)] we obtain, that the application of L-admissible substitutions preserves the validity of proof states. The problem to compute L-admissible substitutions can be solved by using a standard uni cation procedure with occur-check for object variables, and the generic pre x uni cation procedure from [[9],](#_bookmark19) which computes only L-admissible substitutions for position variables, where L is any of the above logics. The notion of valid proof states can subsequently be used to formally prove the L-soundness of any other transformation we might de ne on the signed formula tree. Hence, using signed formula trees gives us a powerful and uniform mechanism to support a variety of logics inside the framework.

# *5 Intuitive and Contextual Reasoning*

*For the design of reasoning and re nement procedures we support an intui-* tive reasoning style. To this end we aim to focus the reasoning process on arbitrary sub-expressions of some actual expression and to have a strong sup- port of contextual reasoning inside a focus. Window inference [[11,6]](#_bookmark16) has been introduced as a formalisation of an intuitive and hierarchical reasoning style.

*The basic objects are windows*

*h[] ` P (a) ) :(a = b)i+*

*which represents that the focus is on the positive (+) formula P (a) ) :(a = b)* (denoted as the content of the window) and the context of this focus is empty ([]). The basic window inference rules are

*(i) to focus on some subexpression of the content of a window*

*h[] ` P (a) ) :(a = b)i+ * *h[a = b] ` P (a)i * *h[a = b] ` aiÆ ;*

*w w*

*where * *denotes a focus derivation step. Note that this rule must be de-*

*w*

*ned for any logical connective (positive ) in the example) and argument* position ( rst argument of ) in the example), in order to adjust the con- text in a consistent manner. While this is not diÆcult for classical logics, where the context of a sub-focus is either equal or an extension of the previous context, this gets diÆcult as soon as we deal with non-classical logics.

*(ii) to transform the content of a window using information from the context*

*h[a = b] ` aiÆ * *h[a = b] ` biÆ ;*

*w*

*(iii) and to leave a focus and return to the parent focus*

*h[a = b] ` biÆ * *h[a = b] ` P (b)i * *h[] ` P (b) ) :(a = b)i+:*

*w w*

*Furthermore, window inference reasoning also allows one to focus on the con-* dition of a window. To this set of basic rules we added a general rule to allow one to introduce a case analysis:

*h[H] ` tip * *h[A j H] ` tip and h[B j H] ` tip;*

*w*

*In order to ensure the soundness of the case analysis* [*2*](#_bookmark8) *the condition h[H] `* A \_ Bi+ is generated as an additional goal. In order to allow for an intui- tive design of reasoning procedures, we integrated window inference reasoning with indexed formula trees (cf. Sec. [4](#_bookmark4)) by annotating positions in the indexed formula trees with windows. As focusing on subexpressions is along the tree structure, opening a sub-window corresponds to adding a window on the re- spective position. Similarly, leaving a focus means removing the window from a position and returning to the previous position.

*The integration of indexed formula trees with windows results in the fol-* lowing structure of the framework: inside the framework, proof states are represented by indexed formula trees with windows. They serve as a sound- ness backbone managing any proof transformation, like e.g. the instantiation of variables or the application of rules from the context. The indexed formula trees are invisible to the reasoning procedures, which communicate with them via the windows annotated to speci c positions and applying window infe- rence rules, which result in a change of the proof state. An example for this is the opening of a sub-window, the instantiation of variables, etc. The diÆ-

*2 Even for intuitionistic logic.*

*culty remains that contextual information must be provided to the reasoning* procedures, like what is the polarity of a window, which are the instantiable variables, and especially which are possible rules from the context of a window and whether they are applicable. These diÆculties are elegantly resolved by using indexed formula trees:

*Polarity of windows: The polarity of a window is simply the polarity of the* position it is attached to in the indexed formula tree.

*Instantiable Variables: The instantiable variables of some window are sim-* ply all variables that are bounded on some -type position in the formula tree. In order to check the admissibility of the instantiation of a variable, we check the L-admissibility of the substitution (i.e. the irre exivity of the new induced ordering , cf. Sec. [4).](#_bookmark4)

*Sampling contextual rules: The context of a window is uniformly deter-* mined by the position in the indexed formula: all parts of the indexed formula, that are related to the actual window by an -type position, are in the logical context. Indeed, according to the reduction rules in Fig. [2,](#_bookmark5) only

*-related formulas may occur together in some nal sequent after apply-*

*ing respective calculus decomposition rules (cf. Sec.* [*4).*](#_bookmark4) *Hence, usable rules* to change the content of a window are for example all negative equations and implications in the context. The application of a negative equation is basically a paramodulation rule, whereas the application of a negative implication is a resolution step. The conditions for such a rule are all the formulas that are -related to this rule. From this we can de ne a uniform concept of a replacement rule ['] u ! v: intuitively, this rule means that we can replace a content u of a window by v, if we prove the conditions '. This notion can be de ned formally as well as if it is admissible wrt. some focus (i.e. position).

*De nition 5.1 [Admissible replacement rules] Let a be a position of la- bel s and polarity p in some indexed formula tree. Then [ ] u ! v is an admissible replacement rule for a, if*

*(i) u and v are either the left- and right-hand side of a negative equation,* that is -related to a, or u and v are -related formulas with the same modal or intuitionistic pre xes, of polarities p and p respectively, and

*-related to a,*

*(ii) and contains all other formulas that are -related to u and v.* *2*

*Application of replacement rules: To apply a replacement rule on some* focus, we must syntactically unify the left-hand side of the rule with the context of the focus as well as their modal pre xes. Furthermore, if the replacement rule is not an equation, the focus and the left-hand side of the rule must have opposite polarities. For uni cation we use standard uni cation procedures for object level variables and rely on the generic pre x uni cation procedure from [[9],](#_bookmark19) in order to compute only substitutions that are admissible for the actual logic. This allows to prove for all logics

*procedure abstract-prove-reachable (window:TLA-States&Transitions)*

*...*

*endprocedure*

*refinement abstract-prove-reachable-refine for abstract-prove-reachable*

*...*

*endrefinement*

*procedure prove-reachable (window:TLA)*

*call-on-abstraction Reachable-Abstraction window abstract-prove-reachable*

*endprocedure*

*Fig. 4. TLA reasoning and re nement procedures considered in* [*[12]*](#_bookmark22) *the soundness of the replacement rule application.*

*In summary, the integration of window inference reasoning with indexed* formulas allows on the one hand for the design of intuitive reasoning proce- dures. On the other hand it provides a simple but powerful mechanism to support contextual reasoning.

# *6 Organising Proof Search*

*The overall proof search inside the framework is organised by the reasoning and* re nement procedures, and some additional language constructs to combine them. The reasoning procedures are based on windows and interact with the proof in two manners: rst, the content of a window can be modi ed by applying a replacement rule from the context of the window. Second, they can introduce a case analysis over some arbitrary formulas A1 \_ ::: \_ An at any stage of the reasoning process. Following the intuitive reasoning paradigm, reasoning procedures can focus the proof search on some sub-window by calling another reasoning procedure on the sub-window. Finally, they can invoke the reasoning on some higher reasoning level by calling a reasoning procedure on some abstraction of the actual window. To this end they indicate the abstraction to be used and the reasoning procedure of the higher reasoning level to call. In case the abstract reasoning procedure succeeds, the re nement of the (abstract) proof is achieved by calling the re nement procedure of the abstract reasoning procedure. Only if the re nement procedure succeeds, the proof planning attempt is successful.

*Example 6.1 Take as an example a reasoning procedure prove-reachable* that is specialised to prove temporal properties of the form 3' (cf. Fig. [4).](#_bookmark9) Hence, the argument of the reasoning procedure is declared to be from the representation TLA. Furthermore let abstract-prove-reachable be the rea-

*soning procedure on the representation TLA-States&Transitions introduced in Sect.* [*2*](#_bookmark1)*, and let Reachable-Abstraction be the abstraction from the repre- sentation TLA to the representation TLA-States&Transitions. In Fig.* [*4*](#_bookmark9) *we sketch the body of prove-reachable: this procedures calls the abstract re- asoning procedure via call-on-abstraction, which abstracts the argument window to the representation TLA-States&Transitions and calls the abstract reasoning procedure abstract-prove-reachable. If this procedure succeeds, the corresponding re nement procedure abstract-prove-reachable-refine is called, which re nes the abstract proof on TLA-States&Transitions to a proof on TLA formulas.* *2*

# *7 Conclusion*

*In this paper we took up the motivation that the task to construct a proof* is an engineering task. This necessitates a methodology to design and apply proof engineering techniques and we emphasised that the techniques developed in proof-planning aim at these goals. Based on that we developed a proof- planning framework, that comes with an explicit notion of di erent reasoning levels. The reasoning levels can be structured via abstraction and re nement relations. Furthermore, we emphasised that cognitive adequate reasoning is a key issue for the design of higher reasoning levels. This entails on the one hand, that the representation of reasoning objects of the di erent reasoning levels should be intuitive. Subsequently, a uniform mechanism to support directly soundness wrt. a large variety of logics has been introduced, without having to alter the representation of logical formulas. Furthermore, we integrated this mechanism with an extension of the window inference reasoning style to allow for the design of intuitive reasoning procedures. A major bene t of this integration is a straightforward and elegant support of contextual reasoning, a key concept for intuitive reasoning. Finally, we sketched how proof-search engineering methods like reasoning procedures, abstractions and re nements can be de ned in the framework, and how these techniques are linked together and subsequently can be applied for proof-search.

*The framework serves as the basis for the ongoing implementation of the* core reasoning mechanisms in the INKA 5.0 system [[1].](#_bookmark11) Future work consists of

*nishing the implementation of the framework, and to validate the adequacy of* the framework by implementing reasoning levels and structuring mechanisms for speci c domains and problems. Theoretical future work is concerned with the integration of further logics into the framework.

# *References*

*[1] Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. System description: INKA 5.0 { a Logic Voyager. In H. Ganzinger, editor, Proceedings of the 16th International Conference on Automated Deduction (CADE), LNAI* *1632, Trento, Italy, 1999. Springer.*

*[2] Christoph Benzmuller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Xiarong Huang, Manfred Kerber, Michael Kohlhase, Karsten Konrad, Erica Melis, Andreas Meier, Wolf Schaarschmidt, Jorg Siekmann, and Volker Sorge.*

*mega: Towards a mathematical assistant. In W. McCune, editor, Proceedings of the 14th International Conference on Automated Deduction (CADE), LNAI* *1249, Townsville, North Queensland, Australia, 1997. Springer.*

*[3] M. Fitting. First-Order Logic and Automated Theorem Proving / 2nd Edition.* *Springer-Verlag New York Inc., 1996. ISBN 0-387-94593-8.*

*[4] Melvin Fitting. Tableau methods of proof for modal logics. Notre Dame Journal* *of Formal Logic, XIII:237{247, 1972.*

*[5] Fausto Giunchiglia and Toby Walsh. A Theory of Abstraction. Journal of Arti cial Intelligence, 56(2-3):323{390, 1992. Also as technical report IRST-* *Technical Report 9001-14.*

*[6] Jim Grundy. Window inference in the HOL system. In Proceedings of the International Workshop on the HOL Theorem Proving System and its* *Applications, 1991.*

*[7] Leslie Lamport. The temporal logic of actions. ACM Transactions on* *Programming Languages and Systems, 16(3):872{923, May 1994.*

*[8] J. Loeckx, H.-D. Ehrig, and M Wolf. Speci cation of Abstract Data Types.* *Teubner, Chichester;New York;Brisbane, 1996. ISBN 3-519-02115-3.*

*[9] Jens Otten and Christoph Kreitz. T-string uni cation: Unifying pre xes in non-classical proof methods. In P Miglioli, U Moscato, and et. al., editors, Proceedings of 5th Workshop on theorem Proving with analytic tableaux and* *related methods, LNAI 1071, pages 244{260. Springer Verlag, 1996.*

*[10] Julian D.C. Richardson, Alan Smaill, and Ian M. Green. System description: proof planning in higher-order logic with -clam. In Claude Kirchner and H el ene Kirchner, editors, Proceedings of the 15th International Conference on* *Automated Deduction (CADE-98), LNAI 1421. Springer, 1998.*

*[11] Peter D. Robinson and John Staples. Formalizing a hierarchical structure of practical mathematical reasoning. In Journal of Logic and Computation,* *volume 3, pages 47{61, 1993.*

*[12] Lincoln Wallen. Automated proof search in non-classical logics: eÆcient matrix proof methods for modal and intuitionistic logics. MIT Press series in arti cial intelligence, 1990.*