Constraint Deduction in an Interval-based Temporal Logic

Jana Koehler & Ralf Treinen

Abstract. We describe reasoning methods for the interval-based modal temporal logic LLP which employs the modal operators *sometimes*, *always*, *next*, and *chop*. We propose a constraint deduction approach and compare it with a sequent calculus, developed as the basic machinery for the deductive planning system PHI which uses LLP as underlying formalism.

1 Motivation

The work presented in this paper was motivated by an application coming from the field of deductive planning. In the PHI system [BBD+93] planning is done on the formal basis of an interval-based modal temporal logic. Apart from plan generation, the reuse and the modification of existing plans is also investigated. Since plan generation and plan reuse are formalized as deductive processes, various proofs in the underlying temporal logic have to be performed raising the need for an efficient proof method for the logic.

In deductive planning, plans are generated by *constructively* proving plan specifications, for example formulae of the form

$$pre \wedge \mathsf{Plan} \rightarrow goals$$

which describe the properties of the desired plan: if Plan is carried out in a situation where the preconditions *pre* hold then the *goals* will be reached.

During the proof, the plan metavariable Plan is replaced by a plan (formula) that satisfies the specification. Plan formulae in LLP [BDK92] provide constructs that allow, e.g. to

- sequentially compose plans from arbitrary subplans or atomic actions,
- incorporate control structures for conditional and iterative plans, and
- abstract from the exact execution time of actions in plans.

When plans are executed, every action leads to a new state of the world, i.e. plans describe temporally ordered sequences of states. This suggests using a modal temporal logic as the underlying formalism for deductive planning and grounding its semantics on *intervals* in contrast to the usual *possible worlds* semantics. Intervals can be seen as possible worlds to which an additional structure has been added, i.e. by considering worlds as sequences of states. This sequential structure of the worlds reflects the semantics of plans.

The interval-based modal temporal logic LLP [BDK92] is consequently developed as a formal basis combining features of *choppy logic* [RP86] with a *temporal logic for programs* [Krö87].

The paper is organized as follows: in Section 2, we describe related work. We review the logic LLP in greater detail and discuss the executability of LLP plans in Section 3. Section 4 describes a sequent calculus approach for deductive planning and plan reuse and discusses its advantages in the underlying context. As an alternative to the LLP sequent calculus we introduce a constraint deduction approach in Section 5. Finally, we conclude with some discussion comparing both approaches in Section 6 and show how the constraint deduction approach is applied for tasks of temporal abstraction.

2 Related Work

Interval-based temporal logics have been proposed by several authors [MM83, Gab89] as appropriate formalisms to describe the behavior of programs or plans. Plans can be decomposed into successively smaller periods or intervals of, e.g. subplans or actions. The intervals provide a convenient framework for introducing quantitative timing details. State transitions can be characterized by properties relating the initial and final values of variables over intervals of time [MM83].

The logic LLP which is considered in this paper is a first order extension of the propositional linear temporal logic PTL(U,X,C) [RP86]. PTL(U,X,C) contains the modal operators *weak-next*, *until* and *chop* and has an interval-based semantics. The concept of *local variables*, the interpretation of which may vary from state to state, was borrowed from [Krö87, MM83, Hal87] in order to describe the action to be performed in a state as well as the effects of the action.

In [Gab89, FO92] a *declarative* as well as an *imperative* view on temporal logic formulae is proposed:

> "A future statement of temporal logic can be understood in two ways: the declarative way, that of describing the future as a temporal extension; and the imperative way, that of making sure that the future will happen the way we want it." (cf. [Gab89],page 402)

Grounded on this view, the logic USF has been developed in [Gab89]. Specification formulae in USF can be automatically re-written into an executable form utilizing an *exec* predicate. For example, *exec*(a_1) will make a_1 true. The re-written formula is an *equivalent* logical formulation of the specification.

Plan specification formulae in LLP can also be viewed as declaratively describing the future as a temporal extension. To obtain a plan, specification formulae are not equivalently transformed but are constructively proved, i.e. an example (plan) is constructed which *satisfies* the specification. The plan can be seen as a program for controlling process behavior: its execution in the initial state is sufficient to reach the specified goals. This view led to the *plans are programs* paradigm which has already been proposed by, e.g. [Bib86, MW87].

In order to benefit from the representational advantages provided by modal logics, reasoning mechanisms for modal formulae have to be developed. Our work fits into the framework of translation oriented methods similar to those described in [Ohl91, FS91]. It extends the constraint deduction approach for serial modal logics with *sometimes* and *always* [FS91] to a non-serial modal temporal logic with the additional modal operators *next* and *chop*.

3 The Interval-based Modal Temporal Logic LLP

LLP (Logical Language for Planning) [BDK92] is an interval-based modal temporal logic which combines features of *choppy logic* [RP86] with a *temporal logic for programs* [Krö87]. The basis of LLP is a many-sorted first order language. Furthermore, we distinguish *local variables*, the value of which may vary from state to state and *global* variables which are the usual logical variables. Local variables are borrowed from *programming logics* where they correspond to program variables.

LLP provides the modal operators $\circ$ (*next*), $\Diamond$ (*sometimes*), $\Box$ (*always*), and the binary modal operator ; (*chop*). Furthermore, control structures like *if-then-else* and *while* are available. In the following we shortly review the main properties of LLP as introduced in [BDK92].

A state σ_i is a valuation assigning domain elements to local variables. Note that only the values of local variables may change from state to state. Function and predicate symbols are *rigid*, i.e. their interpretation does not vary over time. An interval σ is a nonempty finite or infinite sequence of states $\langle\sigma_0\sigma_1\ldots\rangle$. W denotes the set of all intervals. The length of an interval σ is defined as

$$|\sigma| = \begin{cases} \omega, & \text{if } \sigma \text{ is infinite} \\ n, & \text{if } \langle\sigma = s_0 s_1 \ldots s_n\rangle \end{cases}$$

Observe that $|\sigma| = 0$ iff $\sigma = \langle\sigma_0\rangle$ is a *singleton* containing only one state. Intuitively, the length of an interval does not represent the number of states this interval contains, but the number of possible state transitions. The *immediate accessibility* on intervals is defined as the subinterval relationship R with

$$\sigma R \sigma' \text{ iff } \sigma = \langle\sigma_0\sigma_1\sigma_2\ldots\rangle \text{ and } \sigma' = \langle\sigma_1\sigma_2\ldots\rangle.$$

R^* denotes the transitive and reflexive closure of R and R^+ denotes the transitive closure of R. The *composition* is defined as a partial function over the set of intervals W:

$$\sigma \circ \sigma' = \begin{cases} \sigma, \text{ if } \sigma \text{ is infinite} \\ \langle\sigma_0\ldots\sigma_{n-1}\sigma_n\sigma_{n+1}\ldots\rangle, \text{ if } \begin{array}{l}\sigma = \langle\sigma_0\ldots\sigma_{n-1}\sigma_n\rangle \text{ and} \\ \sigma' = \langle\sigma_n\sigma_{n+1}\ldots\rangle\end{array} \end{cases}$$

Global variables are interpreted by mapping them to domain elements using a valuation function. The value of a local variable in an interval σ for a given interpretation is its value in the initial state of the interval. The satisfiability relation $\models$ for modal-free formulae is defined as in classical first order logic. F and T denote the propositional constants *false* and *true*, respectively. For the modal operators we define:

- $\sigma \models_{\mathcal{I}} \circ\phi$ iff $\sigma' \models_{\mathcal{I}} \phi$ for all $\sigma' \in W$ with $\sigma R \sigma'$
- $\sigma \models_{\mathcal{I}} \Diamond\phi$ iff $\sigma' \models_{\mathcal{I}} \phi$ for some $\sigma' \in W$ with $\sigma R^* \sigma'$
- $\sigma \models_{\mathcal{I}} \Box\phi$ iff $\sigma' \models_{\mathcal{I}} \phi$ for all $\sigma' \in W$ with $\sigma R^* \sigma'$
- $\sigma \models_{\mathcal{I}} \phi\,;\psi$ iff there are $\sigma', \sigma'' \in W$, with $\sigma = \sigma' \circ \sigma''$, σ' finite and $\sigma' \models_{\mathcal{I}} \phi$ and $\sigma'' \models_{\mathcal{I}} \psi$

The immediate accessibility relation R is not serial, i.e.

$$\forall \sigma\ \exists \sigma'\ \sigma R \sigma'$$

does not hold since an interval of length zero has no successor. For example, $\circ F$ holds in an interval σ iff σ has length 0, i.e. it is a singleton. More generally, $\circ^n F$ holds in σ iff σ has at most n states, that is iff σ has at most length n-1. A formula $\phi \wedge \neg\circ F \wedge \circ\circ F;\circ\Box\psi$ holds in an interval $\langle \sigma_0 \sigma_1 \sigma_2 \sigma_3 \ldots \rangle$ if

- ϕ holds in the subinterval $\langle \sigma_0 \sigma_1 \rangle$ and
- $\circ\Box\psi$ holds in the subinterval $\langle \sigma_1 \sigma_2 \sigma_3 \ldots \rangle$, i.e., ψ holds in all subintervals $\langle \sigma_n \ldots \rangle$ with $n \geq 2$.

3.1 Properties of LLP

Several results from the literature help to characterize LLP with respect to the expressive power of the modal operators and their axiomatization. The modal operators $\Box$ and $\Diamond$ can be expressed by ; [RP86] using the axioms $\Diamond\phi \leftrightarrow T;\phi$ and $\Box\phi \leftrightarrow \neg\Diamond\neg\phi$.

Further results concern the axiomatization of first order temporal logics. An axiomatization of the propositional linear temporal logic PTL(U,X,C) is developed and a complete and sound decision procedure based on semantic tableaux is given for the logic in [RP86].

Szalas proved in [Sza86] that there is no finistic and complete axiomatization of first order temporal logics of linear and discrete time and gives an infinitary complete proof system for the logic in [Sza87]. In [SH88] it is proven that a first order temporal logic with *equality* and *until* is both weakly and strongly incomplete. A logic is defined to be *weakly incomplete* if "the set of all tautologies (over an arbitrary signature) of the logic is not recursively enumerable or, equivalently, if there is no finistic proof system which is sound and complete for the logic". A logic is *strongly incomplete* if "for no signature the set of tautologies over this signature is recursively enumerable or, equivalently, if the set of tautologies over the empty signature is not recursively enumerable".

The results of Szalas hold for LLP as well with the consequence that there is no sound and complete first order calculus for LLP.

3.2 Planning in LLP

Plan generation is carried out by *constructively* proving plan specification formulae in a sequent calculus [BDK92]. One type of plan specifications used by the PHI system, is *liveness properties* containing in their goal specification a temporally ordered sequence of intermediate subgoal states, e.g. formulae of the form:

$$pre \wedge \mathsf{Plan} \rightarrow \Diamond(goal_1 \wedge goal_2 \wedge \Diamond(goal_3 \wedge \Diamond(goal_4)))$$

As a result of the proof, a plan formula is obtained which is sufficient for the plan specification. The plan utilizes the local variable *ex*, the value of which is a term representing the action to be executed in the current state. An example of a plan formula reads

$$ex = action_1 \wedge \neg\circ F \wedge \circ\circ F\,;\, \Box\phi\,;\, ex = action_2 \wedge \neg\circ F \wedge \circ\circ F\,;\, ex = action_3$$

describing a plan which contains three actions. The subplan containing the actions $action_2$ and $action_3$ can be executed at an arbitrary time after $action_1$ has been executed provided that a formula ϕ always holds between the execution of $action_1$ and $action_2$. ϕ describes in our context the minimal preconditions for the subsequent actions which must be adhered to. This plan formula holds in an interval $\sigma = \langle \sigma_0 \sigma_1 \ldots \rangle$ if

- $ex = action_1$ holds in the first state σ_0 of σ,
- there is an $n \geq 1$ such that all intervals $\langle \sigma_1 \ldots \sigma_n \rangle$ to $\langle \sigma_n \rangle$ satisfy ϕ,
- $ex = action_2$ holds in σ_n, and
- $ex = action_2$ holds in σ_{n+1}.

3.3 Executability of LLP Plans

The PHI planner is able to generate complex plans containing control structures like iteration and case analysis. One prototypical application domain of PHI is a subset of UNIX, namely the mail domain. Here, the planner generates *abstract* plans that are used by a plan recognizer to identify the goals of a user and to offer active help. The following formula shows such an example plan achieving the goal *“Read all mail from sender Joe”*:

$n := 1$;
while $n < length(system\_mbox)$ **do**
    if $sender(msg(n, system\_mbox)) = joe$
      then $ex = type(n, system\_mbox)$
      else $ex = empty\_action$;
$n := n + 1$
od

In order to execute an abstract plan, a *plan interpreter* [Den94] is used, which performs sensing actions in the application system in order to instantiate variables and parameters occurring in the abstract plan. Furthermore, since control structures cannot be executed directly in the application system they are replaced by executable action sequences according to their definition. The plan starts by instantiating the internal counter n with value 1, which refers to the current mail in the mailbox. The while–loop is stepwisely expanded according to the definition of the while-operator:

$$\textbf{while } \varepsilon \textbf{ do } \alpha \textbf{ od} \; ; \beta \;\leftrightarrow\; \textbf{if } \varepsilon \textbf{ then } \alpha \; ; [\textbf{while } \varepsilon \textbf{ do } \alpha \textbf{ od} \; ; \beta] \textbf{ else } \beta$$

The resulting case analysis is resolved by testing the validity of the condition $n < length(system\_mbox)$ in the application system. According to the definition of the if-operator, the non-valid branch is eliminated:

$$\textbf{if } \varepsilon \textbf{ then } \alpha \textbf{ else } \beta \;\leftrightarrow\; [\varepsilon \rightarrow \alpha] \wedge [\neg\varepsilon \rightarrow \beta]$$

Let us assume that the mailbox *system_mbox* of the user contains 3 mails with the first and last from sender Joe. In this situation, the following executable action sequence is obtained as a refinement of the abstract plan:

$ex = type(1, system\_mbox)$;	`read the first mail`
$ex = empty\_action$;	`skip the second mail`
$ex = type(3, system\_mbox)$	`read the last mail`

4 The Sequent Calculus Approach

Together with the logic LLP a sequent calculus was developed which is used by the planner for the constructive proofs of specifications. The sequent calculus extends the S4 sequent calculus (see for example [Wal89]) with rules for the additional modal operators $\circ$ and ; and with derived rules which are of importance for deductive planning and plan reuse.

Typical examples of such rules are the *right*-$\circ$ rule, the *chop composition* and the *sometimes-to-next* rule [BDK92].

- *right*-$\circ$: $\dfrac{\Gamma^* \Rightarrow A, \Delta^*}{\Gamma \Rightarrow \circ A, \Delta}$ with $\Gamma^* = \{B | \circ B \in \Gamma\} \cup \{\Box B | \Box B \in \Gamma\}$, and $\Delta^* = \{B | \circ B \in \Delta\} \cup \{\Diamond B | \Diamond B \in \Delta\}$

- *chop composition*: $\dfrac{\phi_1 \Rightarrow \psi_1 \qquad \phi_2 \Rightarrow \psi_2}{\phi_1 ; \phi_2 \Rightarrow \psi_1 ; \psi_2}$

- *sometimes-to-next*: $\dfrac{\Gamma \Rightarrow \circ\phi \wedge \neg\circ F, \Delta}{\Gamma \Rightarrow \Diamond\phi, \Delta}$

To guide sequent rule applications during a proof, a tactic language is provided in which proof tactics can be described [Den94]. Proof tactics implement search strategies that restrict the search space in the proof and thereby help to reduce unnecessary search effort.

Furthermore, proof tactics specify a certain ordering of modal rule applications. As discussed in [Wal89], the ordering in which modal rules are applied during a sequent derivation influences whether a proof is obtained or not. If an inappropriate order is chosen, a proof may not be found. The cause of this order dependence is the fact that some modal rules lead to formulae being "deleted", i.e. subformulae from the conclusion of a sequent do no appear in the premise, see for example the *right*-$\circ$ rule. Therefore, besides restricting the search space, proof tactics help to maintain sufficient formulae in the sequents to complete the proof.

The use of tactics to guide proofs in the LLP calculus led to an efficient implementation of plan generation and plan reuse in the PHI system [BBD+93]. The tactic language which is provided by the system supports the formulation of new tactics when new proof tasks must be solved by the system or when the syntactical class of formulae, on which these proofs are performed, is extended.

Practical experiences, e.g. comparing the effort for plan modification to plan generation [Koe94b] show that proof tactics lead to a very efficient reasoning in the underlying logic. Hence, proof tactics can be seen as one way of developing efficient reasoning methods for modal logics based on Gentzen type calculi.

On the other hand, such classical calculi for modal logics and its extensions, e.g. temporal logic and dynamic logic, which are usually of Hilbert, Gentzen or Tableaux type are often criticized for their inefficiency because "the branching rate in the search space is very high" and because they " require special implementations of deduction systems" (cf. [Ohl91]).

Alternatively, a translation of modal formulae into predicate logic syntax such that standard predicate logic deduction systems are applicable has been proposed by some authors [Ohl91, FS91] for modal logics with possible worlds semantics which provide the modal operators $\Diamond$ and $\Box$.

In the work presented in [Ohl91] a *functional* and a *relational* translation method are developed. The target calculus for the formulae obtained by the translation is the resolution calculus. The relational translation introduces special predicates representing the accessibility relations. This approach is very flexible because different kinds of accessibility relations can easily be handled but the number and the size of translated clauses is increased by the literals which are necessary to represent the possible worlds. Since standard resolution strategies do not differentiate between "normal" and "special" predicates, many unnecessary resolution steps may occur.

To overcome the problem, the functional translation method has been developed whereby the relevant information about worlds is represented in terms, and reasoning about possible worlds is done with specialized unification algorithms. On one hand, this method leads to a more efficient calculus, while on the other hand, it is not known whether this method can be extended to more expressive modal logics, which, for example, contain the *chop* operator.

A generalization of the relational translation method, restricted to modal logics with serial accessibility relations, is presented in [FS91]. A translation into a first order constraint logic is proposed and a hybrid reasoning method combining ordinary deduction with special purpose methods for constraint processing is developed.

During the development of the sequent calculus approach for LLP the question arose whether these translation oriented methods can be extended in order to provide a framework for a semantics-based translation of LLP.

Furthermore, when such a translation of LLP into first order logic is possible, the question arises as to how an efficient calculus can be developed which might serve as an alternative to the sequent calculus approach. It turns out that this remains a difficult problem even for the translated logic.

5 The Constraint Deduction Approach

In the previous section we described the motivation that led to the development of a semantics-based translation method for LLP. Since the logic we consider is more expressive than those logics for which translation oriented methods already exist [FS91, Ohl91] we hope to gain further insight into how far these results generalize.

A relational translation is developed during which special predicate symbols are introduced representing the accessibility relations on intervals. Following the method of [FS91], we proceed in three steps in order to obtain a constraint theory and a set of constrained clauses. These can be proven, for instance, by constrained resolution [Bür91], which can be realized by a resolution prover and a satisfiability checker for constraints. The steps are

1. translation of the modal logic LLP into a constraint first order logic CPL by reification of the intervals,
2. transformation of CPL formulae into constrained prenex normal form and
3. skolemization of formulae in constrained prenex normal form.

5.1 The Translation into CPL

The translation of LLP formulae into the constraint predicate logic CPL transforms the modal-logic features of LLP into predicate logic by reification of the intervals. For this purpose we first transform the signature Σ_M of LLP into the signature Σ_P of CPL as follows:

1. CPL contains the two sorts D, denoting the domain of LLP, and W, which denotes the set of intervals.
2. The domain variables in CPL are the global variables from LLP.
3. The function and predicate symbols from LLP carry over to CPL. Observe that, in contrast to [FS91], functions and predicates have a fixed interpretation in all intervals and therefore do not have to be equipped with an extra argument for the actual interval.
4. The local variables of LLP, which may change the interpretation from interval to interval, are translated into unary function symbols of type $(W \rightarrow D)$.
5. For the translation of modal operators we need the additional predicate symbols S (type W), $\geq$ (type WW), $\gg$ (type WW) and $\oplus$ (type WWW). Atoms constructed with these predicates are called *constraints*, we write them in mixfix notation Sx, $x \geq y$, $x \gg y$ and $x = y \oplus z$.

An LLP interpretation $\mathcal{I}_M$ is translated into a CPL interpretation $\mathcal{I}_P$ as follows: The domain of $\mathcal{I}_P$ consists of the domain of $\mathcal{I}_M$ (for sort D) and the set of all intervals (for sort W). The interpretation of global variables and function symbols as well as predicate symbols remains unchanged. The interpretation of a function symbol translating a local variable x is the function that maps an interval σ to the value of x in σ, which is its value in the initial state of σ. The interpretation of the predicate symbol S is the set of all intervals of length 0, $(x, y) \in \gg^{\mathcal{I}_P}$ iff xRy, $(x, y) \in \geq^{\mathcal{I}_P}$ iff xR^*y and $(x, y, z) \in \oplus^{\mathcal{I}_P}$ iff $x = y \circ z$.

As in [Ohl91], we define a translation function π that takes an LLP formula and translates it into a CPL formula. Since intervals are reified in CPL, the actual interval will be explicitly represented in the translated formula. Therefore, π takes as its second argument the variable which refers to the actual interval. The following translation rules from LLP into CPL are used by π:

$$
\begin{aligned}
\pi[x, w] &:= x \quad \text{if } x \text{ a global variable} \\
\pi[x, w] &:= x(w) \quad \text{if } x \text{ a local variable} \\
\pi[f(t_1, \ldots, t_n), w] &:= f(\pi[t_1, w], \ldots, \pi[t_n, w]) \\
\pi[P(t_1, \ldots, t_n), w] &:= P(\pi[t_1, w], \ldots, \pi[t_n, w]) \\
\pi[(F \wedge G), w] &:= \pi[F, w] \wedge \pi[G, w] \\
\pi[(F \vee G), w] &:= \pi[F, w] \vee \pi[G, w] \\
\pi[\neg F, w] &:= \neg\pi[F, w] \\
\pi[\exists x F, w] &:= \exists x\, \pi[F, w] \\
\pi[\forall x F, w] &:= \forall x\, \pi[F, w] \\
\pi[\Diamond F, w] &:= \exists v(w \geq v \wedge \pi[F, v]) \\
\pi[\Box F, w] &:= \forall v(w \geq v \rightarrow \pi[F, v]) \\
\pi(\circ F, w] &:= \forall v(w \gg v \rightarrow \pi[F, v]) \\
\pi[(F;G), w] &:= \exists v, u(w = v \oplus u \wedge \pi[F, v] \wedge \pi[G, u])
\end{aligned}
$$

Theorem 1 states the soundness of the translation:

Theorem 1 *For all LLP interpretations $\mathcal{I}_M$, LLP formulae ϕ, W-variables w holds*

$$\mathcal{I}_M \models \phi \Leftrightarrow \mathcal{I}_P \models \forall w\, \pi[\phi, w]$$

where $\mathcal{I}_P$ is the translation of $\mathcal{I}_M$.

Proof sketch: The soundness of the translation can be shown by structural induction over terms and formulae. The base case for global variables is trivial because the assignment of global variables does not differ between $\mathcal{I}_M$ and $\mathcal{I}_P$. Proving the soundness of the translation rules for function and predicate symbols is trivial because they are rigid and thus their interpretation remains unchanged. The correctness of the translation rule for local variables follows from the definition of $\mathcal{I}_P$: the interpretation of the function symbol translating a local variable is the value of that variable in the first state of an interval. The cases for normal (non-modal) connectives are straightforward, their interpretations remains unchanged. To prove the correctness of the translation rules for modal operators the LLP semantics as well as the definition of the semantics of the resulting constraint predicates is exploited. ■

As a short example the translation of the formula

$$T;P(v) \rightarrow \Diamond P(v)$$

where v is a local variable is shown below. This formula is valid; it is in fact a consequence of the following theorem of LLP:

$$T;\phi \leftrightarrow \Diamond\phi$$

After expanding $\rightarrow$, we obtain ($i_1, i_2, \ldots$ are W-variables):

$$\begin{aligned}
&\pi[\neg\Big(T;P(v)\Big) \vee \Diamond P(v), i_1] \\
&= \neg\pi[T;P(v), i_1] \vee \pi[\Diamond P(v), i_1] \\
&= \neg\exists i_2, i_3\Big(i_1 = i_2 \oplus i_3 \wedge \pi[T, i_2] \wedge \pi[P(v), i_3]\Big) \vee \pi[\Diamond P(v), i_1] \\
&= \neg\exists i_2, i_3\Big(i_1 = i_2 \oplus i_3 \wedge \pi[T, i_2] \wedge \pi[P(v), i_3]\Big) \vee \exists i_4\Big(i_1 \geq i_4 \wedge \pi[P(v), i_4]\Big) \\
&= \neg\exists i_2, i_3\Big(i_1 = i_2 \oplus i_3 \wedge T \wedge P(v(i_3))\Big) \vee \exists i_4\Big(i_1 \geq i_4 \wedge P(v(i_4))\Big)
\end{aligned}$$

5.2 The Transformation into Prenex Normal Form

The formulae computed by the above translation mechanism have the property that all constraints τ occur only in the form $\forall\bar{x}\tau \rightarrow \psi$ or $\exists\bar{x}\tau \wedge \psi$. The aim is to preserve this property during transformation into prenex normal form as this will allow the translation into constrained clauses in the last step. Rules for transforming into prenex normal form maintaining the above property have been given in [Fri91]:

$$\neg\exists\bar{x}(\tau \wedge \phi) \Rightarrow \forall\bar{x}(\tau \rightarrow \neg\phi)$$
$$\exists\bar{x}(\tau \wedge \phi) \wedge \psi \Rightarrow \exists\bar{x}(\tau \wedge \phi \wedge \psi)$$

provided that ψ does not contain variables of $\bar{x}$.[1] The problem is that the transformation

$$\exists\bar{x}(\tau \wedge \phi) \vee \psi \quad \Rightarrow \quad \exists\bar{x}(\tau \wedge [\phi \vee \psi]) \tag{7.1}$$

needs additional conditions in order to be an equivalence transformation. This rule was proven correct in [Fri91] for $\bar{x}$ consisting of the single variable x, τ containing the variable x only and τ being satisfiable in every model under consideration. In [FS91], binary constraints $K(x, y)$ have been considered where y is bound by the surrounding existential quantifier but x is free. There, the correctness condition was that K, denoting the accessibility relation between worlds, is serial.

Generally speaking, (7.1) is an equivalence transformation in the class $\mathcal{C}$ of models if ψ does not contain x and if

$$\mathcal{C} \models \tilde{\forall}\exists\bar{x}\,\tau \tag{7.2}$$

[1] Only rules for $\exists$ are considered here; the rules for $\forall$ being dual.

holds where $\tilde{\forall}\gamma$ denotes the universal closure of a formula γ. As the reader easily verifies, condition (7.2) subsumes those of [Fri91] for sorted deduction and of [FS91] for reasoning with a serial accessibility relation.

While looking at the constraints introduced by the translation process, it turns out that rule (7.1) applies to the quantifiers which arise from the operators ; and $\Diamond$, as for all CPL interpretations $\mathcal{I}_P$

$$\mathcal{I}_P \models \forall x\, \exists y\, x \geq y, \text{ and}$$
$$\mathcal{I}_P \models \forall x\, \exists y, z\, x = y \oplus z.$$

The situation is different with a formula

$$\exists y \Big(x \gg y \wedge \phi \Big) \vee \psi \tag{7.3}$$

as R is not serial. Two cases have to be considered: if the value of x is a sequence of length 0, then (7.3) is obviously equivalent to ψ. If the value of x has a length of at least 1, then we can replace (7.3) by $\exists y(x \gg y \wedge [\phi \vee \psi])$, or alternatively since $\gg$ is functional, by $\forall y(x \gg y \rightarrow [\phi \vee \psi])$. Note that any occurrence of a formula 7.3 is within the scope of a quantifier for x. The case distinction is done by splitting the quantifier for x by using the rule

$$\exists x \gamma \quad \Rightarrow \quad \exists x \Big(Sx \wedge \gamma \Big) \vee \exists x \Big(\neg Sx \wedge \gamma \Big) \tag{7.4}$$

and accordingly for a universal quantifier. As an example, the following formula is transformed into prenex normal form:

$$\forall i_1 \Big\{ \exists i_2 \Big(i_1 \gg i_2 \wedge P(v(i_2)) \Big) \vee Q(w(i_1)) \Big\}$$

We split the quantifier for i_1 using the dual of rule (7.4):

$$\begin{gathered} \forall i_1 \Big\{ Si_1 \rightarrow \exists i_2 \Big(i_1 \gg i_2 \wedge P(v(i_2)) \Big) \vee Q(w(i_1)) \Big\} \\ \wedge\; \forall i_1 \Big\{ \neg Si_1 \rightarrow \exists i_2 \Big(i_1 \gg i_2 \wedge P(v(i_2)) \Big) \vee Q(w(i_1)) \Big\} \end{gathered}$$

The prenex normal form is arrived at after three further transformation steps:

$$\begin{aligned} &\forall i_1 \Big\{ Si_1 \rightarrow Q(w(i_1)) \Big\} \;\wedge\; \forall i_1 \Big\{ \neg Si_1 \rightarrow \exists i_2 \Big(i_1 \gg i_2 \wedge (P(v(i_2)) \vee Q(w(i_1))) \Big) \Big\} \\ &\forall i_1, j_1 \Big\{ Si_1 \wedge \neg Sj_1 \rightarrow Q(w(i_1)) \wedge \exists j_2 \Big(j_1 \gg j_2 \wedge (P(v(j_2)) \vee Q(w(j_1))) \Big) \Big\} \\ &\forall i_1, j_1 \Big\{ Si_1 \wedge \neg Sj_1 \rightarrow \exists j_2 \Big(j_1 \gg j_2 \wedge Q(w(i_1)) \wedge (P(v(j_2)) \vee Q(w(j_1))) \Big) \Big\} \end{aligned}$$

5.3 The Skolemization

The rules for the skolemization of formulae in constrained prenex normal form can be found in [Fri91]. In the example of the last subsection, the binary skolem function symbol *sk* is associated with the existentially quantified variable i_2. We obtain the constrained formula

$$Q(w(i_1)) \wedge (P(v(sk(i_1, j_1))) \vee Q(w(j_1))) \quad / \quad Si_1 \wedge \neg Sj_1$$

as well as the sentence

$$\forall i_1, j_1 \left(Si_1 \wedge \neg Sj_1 \rightarrow j_1 \gg sk(i_1, j_1) \right)$$

which in this case constitutes the *constraint theory*.

5.4 Solving Constraints

In order to obtain a hybrid reasoning system [Fri91] for CPL formulae a constraint solver is needed which can decide the satisfiability of conjunctions of the constraint atoms in a given constraint theory. The constraint theory employed here unfortunately turns out to be undecidable. This can be shown quite easily using the method of [Tre92]. Furthermore, the undecidability of the satisfiability of the constraints follows from the fact that validity of LLP-formulae is undecidable, while constrained resolution is complete relative to the satisfiability of the constraints [Bür91].

Among the positive results on decidability of related constraint systems we mention the seminal paper by Rabin [Rab69], where a decision procedure for the monadic second order theory of strings is given. As a corollary of [Rab69], the full first order theory of *finite* intervals with the predicates $\geq$ and $\gg$ (but without $\oplus$) is decidable.

5.5 Application of the Translation Method

A translation of temporal-logic formulae into first-order logic by reification of intervals leads to an explicit representation of the temporal information contained in the formulae. This property can be used to explicitly reason about temporal relationships. In Section 3 we showed the following example of a plan specification formula in form of a liveness property:

$$pre \wedge \mathsf{Plan} \rightarrow \Diamond(goal_1 \wedge goal_2 \wedge \Diamond(goal_3 \wedge \Diamond(goal_4)))$$

Translating the goal specification formula into a CPL formula and separating the constraints from the first-order part of the formula leads to

$$\begin{gathered} \exists i_1, i_2, i_3 \;\; i_1 \geq i_0 \;\wedge\; i_2 \geq i_1 \;\wedge\; i_3 \geq i_2 \wedge \\ goal_1(i_1) \wedge goal_2(i_1) \wedge goal_3(i_2) \wedge goal_4(i_3) \end{gathered}$$

Temporal abstraction of plan specifications can now be implemented by removing constraint formulae and thus relaxing the temporal constraints that are specified for a set of subgoals. In the above mentioned example, we require the plan to achieve $goal_1$ and $goal_2$ in the same interval, while $goal_3$ and $goal_4$ have to be achieved later. When we remove the set of constraints, no ordering of subgoal states is required anymore. Temporal abstraction grounded on a manipulation of temporal constraints is used in the PHI system to index plans in a plan library [Koe94a]. This method leads to a well-defined temporal abstraction process and allows to prove a *monotonicity property* that holds between original and abstracted plan specifications. It states that an existing subset relationship between the set of models satisfying two LLP plan specifications is preserved as a subset relationship between the set of models satisfying the abstracted CPL formulae. This property ensures that a retrieval method can be developed that finds a plan solving a given plan specification whenever it exists in the plan library.

6 Conclusion

Two calculi for an interval-based modal temporal logic are discussed in this paper: a sequent calculus developed in [BDK92] and a constraint deduction approach. The sequent calculus was implemented as the basis for deductive planning and plan reuse in SICSTUS PROLOG. First practical experiences demonstrated that the sequent calculus approach provides an efficient reasoning method when proofs are guided by tactics. The tactics support the declarative representation of control knowledge which helps to keep the search space to a manageable size. A tactic language is used to describe the tactics and makes it easy to develop and incorporate new tactics into the system.

A translation into constraint predicate logic is presented as an alternative approach. In this case, the undecidability of LLP reflects in the undecidability of constraint satisfiability, although the basic machinery of constrained resolution itself is known to be complete. This localization of the undecidability in the constraint part raises the hope of finding decidable fragments of LLP by isolating decidable fragments of the constraint theory.

Acknowledgements

We would like to thank Susanne Biundo, Hans-Jürgen Bürckert, Dietmar Dengler, Hans-Jürgen Ohlbach and Gert Smolka for their interest and for fruitful discussions.

References

[Aba89] Martin Abadi. The power of temporal proofs. *Theoretical Computer Science*, 65:35–83, 1989.

[Bau92] Mathias Bauer. An interval-based temporal logic in a multivalued setting. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE'11)*, LNCS 607, pages 355–369, Saratoga Springs,NY,USA, 1992. Springer.

[BBD+93] Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, and Gabriele Paul. PHI - a logic-based tool for intelligent help systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

[BDK92] Susanne Biundo, Dietmar Dengler, and Jana Koehler. Deductive planning and plan reuse in a command language environment. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 628–632, 1992.

[Bib86] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Bür91] Hans-Jürgen Bürckert. *A Resolution Principle for a Logic with Restricted Quantifiers*. Lecture Notes in Artificial Intelligence, vol. 568. Springer, 1991.

[Den94] Dietmar Dengler. An adaptive deductive planning system. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 610–614, Amsterdam, NL, August 1994. John Wiley & Sons.

[FO92] Michael Fisher and Richard Owens. From the past to the future: Executing temporal logic programs. In A. Voronkov, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, pages 369–380. Springer, Berlin, Heidelberg, 1992.

[Fri91] Alan M. Frisch. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

[FS91] Alan M. Frisch and Richard R. Scherl. A general framework for modal deduction. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 196–207, Cambridge,Massachusetts, 1991. Morgan Kaufmann.

[Gab89] Dov Gabbay. Declarative past and imperative future: Executable temporal logic for imperative systems. In H. Barringer and A. Pnueli, editors, *Proceedings of the Colloquium on Temporal Logic in Specification 1987*, LNCS 398, pages 402–450, Altrincham, 1989. Springer.

[Hal87] Roger Hale. Temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 91–119. Academic Press, 1987.

[Koe94a] Jana Koehler. An application of terminological logics to case-based reasoning. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International*

Conference on Principles of Knowledge Representation and Reasoning, pages 351–362. Morgan Kaufmann, San Francisco, CA, 1994.

[Koe94b] Jana Koehler. Avoiding pitfalls in case-based planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, pages 104–109, Chicago, IL, 1994. AAAI Press, Menlo Park.

[Krö87] Fred Kröger. *Temporal Logic of Programs*. Springer, Heidelberg, 1987.

[MM83] Ben Moszkowski and Zohar Manna. Reasoning in interval temporal logic. In E. Clarke and D. Kozen, editors, *Proceedings of the Conference on Logics of Programs*, LNCS 164. Springer, 1983.

[MW87] Zohar Manna and Richard Waldinger. How to clear a block: Plan formation in situational logic. *Journal of Automated Reasoning*, 3:343–377, 1987.

[Ohl91] Hans-Jürgen Ohlbach. Semantics-based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–775, 1991.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[RP86] Roni Rosner and Amir Pnueli. A choppy logic. In *Symposium on Logic in Computer Science*, Cambridge, Massachusetts, 1986.

[SH88] Andrzej Szalas and Leszek Holenderski. Incompleteness of first-order temporal logic with until. *Theoretical Computer Science*, 57:317–325, 1988.

[Sza86] Andrzej Szalas. Concerning the semantic consequence relation in first-order temporal logic. *Theoretical Computer Science*, 47:329–334, 1986.

[Sza87] Andrzej Szalas. A complete axiomatic characterization of first-order temporal logic of linear time. *Theoretical Computer Science*, 54:199–214, 1987.

[Tre92] Ralf Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5):437–457, November 1992.

[Wal89] Lincoln A. Wallen. *Automated Deduction in Nonclassical Logics*. MIT Press, Cambridge, London, 1989.