Deductive Formation of Recursive Workflows

Jeremy Forth
IWE Consulting
Santa Clara
California, US
jforth@iweng.org

Richard Waldinger
SRI International
Menlo Park
California, US
waldinger@ai.sri.com

Abstract

In this work, we present an action theory with the power to represent recursive plans and the capability to reason about and synthesize recursive workflow control structures. In contrast with the software verification setting, reasoning does not take place solely over predefined data structures, and neither is there a process specification available in recursive form. Rather, specification takes the form of goals, and domain structure takes the form of a physical application setting containing objects. For this reason, well-founded induction is employed for its suitability for practical action domains where recursive structures must be described or inferred. Under this method, termination of the synthesized recursive workflow is a property that follows automatically.

We show how a general workflow recursive construct is added to an action language which is then augmented with induction. This formalism is then transformed in a way amenable to automated reasoning. We then demonstrate the method with a particular example specified in the theory, and then extracted from a proof constructed by the SNARK first order theorem prover.

Introduction

Due to the power and succinctness of recursive representations of processes, substantial value is offered by the inclusion of this control structure in workflow languages. Accordingly, almost all present workflow execution engines employ some type of facility to represent executional repetition.

However, a main application of workflows is modeling processes in real world settings, and it is therefore important to be able to reason about these workflows' consequences, properties and formation. Prior work has applied action theories to reasoning about workflows, yet has been limited in expressivity to linear conditional execution paths. As a consequence, the underlying workflow language is sub-Turing-complete, with a consequential difficulty in expressing many commonly occurring practical processes.

This work sets out to introduce recursive process expressivity into workflow languages to allow reasoning for verification of suitable properties prior to the deployment of a workflow, and also synthesis of a workflow that satisfies user-specified properties.

The workflow setting differs from that of software verification because software verification specification is achieved by recourse to a recursive specification written in terms of pre-defined data structures. By contrast, workflow specification is made through a set of goals the workflow is expected to achieve in its environmental setting.

Reasoning about workflows is thus more akin to reasoning about action domains from AI than of a software verification or synthesis task. Accordingly, the foundational method chosen here for reasoning about recursive control structures is well-founded induction. Well-founded induction is a very general induction principle that implies most other induction rules as special cases. It affords the ability to define which domain elements constitute a well-founded structure. It also offers the potential to automatically dynamically infer a well-founded structure from the arrangement of a domain.

Reasoning about action domains has in the past focused on a linear (possibly conditional) representation of the plan. Many real world domains however are structured in ways that prevent the quantity or extent of a domain entity being known at process formation time. Providing for these contingencies using a linear plan either quickly becomes prohibitive in plan space, or is theoretically impossible. A recursive description however, permits the representation of a plan tolerant of the domain's contingencies in a tractable plan space. Depending on the contingency's type, recursive expressivity has the potential to reduce a linear growth in space to constant space. In other cases, contingency types that cause an exponential growth in plan space may be reduced by recursion to grow in linear space.

In this paper, we formulate an approach to the representation and reasoning of recursive workflows by using an action theory to represent the workflow for the purposes of identifying consequences of execution, and also the synthesis of a workflow in response to a given set of conditions. There are two main aims. The first is to derive the known consequences of a recursive workflow, and the second to synthesize a workflow which may incorporate recursion.

Motivating Example

Let us assume we have a bag containing a number of books for return to the library. The quantity of books, although known at the end of plan execution, is indeterminate during plan construction, and usually during execution also.

There are several alternative means for representing a plan guaranteeing all the books are returned. If an upper bound

on book quantity is known in advance, then it becomes possible to formulate an exhaustive conditional plan. The plan can guarantee it can cater for the worst-case quantity of books. However, even with this assumption, the method lacks feasibility because we must always generate contingencies for the worst case. Frequently, worst-case conditions are not known in advance, and therefore an upper bound assumption is unrealistic in many domains.

Assume we have a set of library books, *book-set*$_0$, and we wish to return all the books in the set to the library. The primitive action we can execute, however, is *return-one*(*book-set*), which returns only a single book from its argument, *book-set*, by placing it in the return box. We also provide a test operation, *empty-set*(*book-set*), which determines if its argument set is empty, indicating when all the books have been deposited.

We attempt to construct a plan *return-all*(*book-set*$_0$) that will return all the books in its argument *book-set*$_0$. It is specified to satisfy the goal condition P1.

$$\forall(\textit{book-set}_0)\,\exists(z)\,\forall(s0) \quad (s_0 \lhd z) \blacktriangleright \textit{empty-set}(\textit{book-set}_0) \tag{P1}$$

This is to say that for any input set *book-set*$_0$, we attempt to show the existence of a composite action z that, when performed in any initial situation s_0, will have the effect of emptying *book-set*$_0$.

In this example, to meet the goal of returning all books, the plan should express that we repetitively remove each book individually from the bag, and place it in the book drop, and stop only when the bag is empty.

We will approach the problem here by automatically synthesizing a recursive plan that solves the problem in the general case (P2).

$$\begin{aligned}\textit{return-all}(\textit{book-set}_0) \Leftarrow\ & \textit{if } \textit{empty-set}(\textit{book-set}_0),\\ & \textit{no-op},\\ & \textit{return-one}(\textit{book-set}_0)\,;\\ & \textit{return-all}(\textit{book-set}_0)\end{aligned} \tag{P2}$$

The system will be provided with information about the causal effect of an action for placing *one book* into the return bin. We also expect to give information about the structure of the domain sufficient for the system to know that it will finish the task.

In support of this, information will be given to justify the use of well-founded induction. This will be achieved through the attachment of the domain scenario to a general well-founded relation. Here, the book bag will be regarded as a finite set allowing the proper subset relation between sets to be a well-founded relation underpinning induction.

An Action Theory for Workflows

To represent and reason about workflows, we develop an action theory having its roots in a version of Plan Theory (Manna and Waldinger 1987), a logical action formalism that makes use of an explicit state (of the world) representation. Plan Theory is in turn a version of situation calculus.

Process Plan Theory (PPT) as developed here is written in a multi-sorted predicate calculus with equality, with

- a sort $\mathcal{O}$ for *objects* (variables $t, t_0, t_1, \ldots$),
- a sort $\mathcal{S}$ for *situations* (variables $s, s_1, s_1, \ldots$),
- a sort $\mathcal{E}$ for *events* (variables $z, e, e_0, e_1, \ldots$), and
- a sort $\mathcal{P}$ for *propositions* (variables $p, p_0, p_1, \ldots$).

A number of foundational functions and relations are also used.

- To capture the result of an event, the function $s \lhd e$ is introduced from a situation and event to the new situation resulting from the execution of the event.
- To represent the action sequence e_1 followed by e_2 the function $e_1 \circ e_2$ from an event and event to event is introduced to represent the sequential composition of e_1 and e_2.
- To represent the value of a term t in a particular situation, the function $s \rhd t$ from a situation and term to an object is introduced.
- To represent that a proposition p holds in situation s, the relation $s \blacktriangleright p$ is introduced.

Within the language, all objects named by constants $\{\tau_0, \tau_1, ...\}$, all situations $\{\sigma_0, \sigma_1, ...\}$, all events $\{\epsilon_0, \epsilon_1, ...\}$, and all propositions $\{\phi_0, \phi_1, ...\}$ will be assumed to be uniquely named. Axiom P4 making use of constructs from (Baker 1989) guarantees this property.

$$\begin{aligned}&\textit{UNA}\{\tau_0, \tau_1, ...\} \ \wedge\ \textit{UNA}\{\sigma_0, \sigma_1, ...\} \ \wedge\\ &\textit{UNA}\{\phi_0, \phi_1, ...\} \ \wedge\ \textit{UNA}\{\epsilon_0, \epsilon_1, ...\}\end{aligned} \tag{P4}$$

Propositions, expressions and relations are reified to permit us to quantify over these objects, and we will assume implicit universal quantification for free variables. The first axioms for PPT are identity axioms for situations and events in P5 and P7 respectively.

$$(s \lhd \textit{no-op}) = s \tag{P5}$$

$$(\textit{no-op} \circ e) = e \tag{P6}$$

$$(e \circ \textit{no-op}) = e \tag{P7}$$

Associativity of action terms is inherent in the understanding of the plan language, and is stated explicitly in P8. Composition of events are described by the equality P9 between function terms.

$$(e_1 \circ e_2) \circ e_3 = e_1 \circ (e_2 \circ e_3) \tag{P8}$$

$$s \lhd (e_1 \circ e_2) = (s \lhd e_1) \lhd e_2 \tag{P9}$$

Conditional events are captured by the following two axioms, P10 for the conditional action with a positive condition test result, and P11 for a negative condition test result.

$$(s \lhd \mathit{if}(p_0, e_1, e_2)) \blacktriangleright p \leftarrow (s \blacktriangleright p_0 \wedge (s \lhd e_1) \blacktriangleright p)) \quad \text{(P10)}$$

$$(s \lhd \mathit{if}(p_0, e_1, e_2)) \blacktriangleright p \leftarrow (\neg(s \blacktriangleright p_0) \wedge (s \lhd e_2) \blacktriangleright p)) \quad \text{(P11)}$$

Domain initial conditions are represented as a conjunction of axioms which may take a range of forms. P12 represents a simple case that proposition ϕ holds in the initial situation s_0.

$$(s_0 \blacktriangleright \phi) \quad \text{(P12)}$$

The action theory here is capable of utilizing existing solutions to the frame problem, such as non-monotonic completion or successor-state axioms (Reiter 1991). However, since this is orthogonal to the subject of the present work, we refer the interested reader to (McCarthy and Hayes 1969; Shanahan 1997). We will refer to P5 to P12 together as ProPT.

Projection and Planning

The action theory of the last section may be employed in two main modes of reasoning. The first, consequence projection, determines the new state of the domain after a series of events have taken place. Consequences of the type P13 may for example be used with the theory ProPT to test for conditions that hold following the execution of a plan.

$$\exists p \forall s_0 . (s_0 \lhd \text{plan}) \blacktriangleright p \quad \text{(P13)}$$

The second mode of reasoning, plan synthesis, discovers a plan of events that will cause the domain to have the conditions specified in the goal. Parameterized plans may take zero or more argument expressions as input. Here, we will consider the case of one argument denoted by a. This may for instance be the name of an object to be moved, or the name of a bank account to which transfers are directed.

For the input expression variable a, we would like to find a composite event z that makes the goal condition (a relation on a) true for any initial situation s_0. This is equivalent to proving a goal taking the form P14.

$$\forall a \exists z \forall s_0 \, . \, \Gamma(s_0, a, z) \quad \text{(P14)}$$

Representation of Recursion

The technique of accomplishing a task by repeated application of a sub-process has been shown over time to be almost universal in its use. The reason lies in the practical advantages that accrue with the succinctness of representation made possible with a recursive (or to a lesser extent iterative) description. While for smaller examples where the domain is fully specified a priori, it may be possible to represent the plan in linear form (where recursive calls are unwound), but for larger problems and problems that are incompletely specified, it becomes impossible to represent the plan in this flattened form.

Whenever a workflow acts on a domain whose condition is not fully known at the time of workflow construction, its actions must be contingent on the domain's state (e.g. quantity) at the point of invocation. The chief reason for the succinctness afforded by recursion is that it introduces parameterized sub-processes that allow for a conditional invocation to more efficiently represent this contingent repetitive activity.

In PPT, we attempt to prove the goal P14. Within the plan term z, we now introduce the function f to represent the recursive process, and introduce a the input parameters passed to it on the first call. $f(a)$ will then represent the first invocation of the recursive process.

Recursive processes may be executed or simulated using the reasoning apparatus we have so far. To reason about properties of recursive plans, however, the underlying reasoning system must incorporate induction.

Reasoning about Recursive Plans: Induction

Recursion and induction are closely linked. We are justified in employing inductive reasoning solely because the domain structures under consideration are recursively defined (or alternatively, are assumed to be recursively defined).

Proving properties of programs by *structural induction* was suggested by (Burstall 1969), but (Floyd 1967) also used a related approach for proving properties of flowchart processes where the induction principle appeared implicitly. In (Clark and van Emden 1981) flowcharts are described by logic programs. By virtue of the equivalence of a logic program's minimal model and its least fixed point, consequence verification may be viewed in terms of fixed point induction (Park 1970) and also structural induction as used by (Boyer, R. S. and Moore, J S. 1979).

The technique of structural induction makes use of the constructor (or destructor) functions from a recursive definition to formulate the hypothesis of an induction rule. It may be considered an instantiated form of (and derived from) the more general well-founded induction. While structural induction yields termination properties, it must be custom built with a constructor (or destructor) function for each individual proof.

With well-founded induction, the choice of a suitable ordering relation in conjunction with a suitable induction hypothesis results in an intuitive proof of goal properties, while also conveniently yielding termination properties. Termination of a recursive routine occurs whenever there is an absence of an infinite sequence of recursive calls. This condition is guaranteed by the well-foundedness requirement of the induction used for the proof of properties (and synthesis). If the induction is well-founded, then there can be no infinite sequence of recursive calls. A consequence of the property of well-foundedness is a necessary and sufficient condition for the existence of a minimal element of any nonempty set of elements.

In many prior works, e.g. (Kraan, Basin, and Bundy 1993), it has been usual practice to precompile a set of derived induction rules which are then selected according to the problem. The rules apply to particular recursive data

types such as natural numbers, sets and lists. Here, however we will use well-founded induction and equip it with a well-founded relation that enables the proof of the desired goal to be completed. Due to the generality of well-founded induction, we need only one induction rule.

Workflow domains are frequently comprised of real world structural objects, in contrast with software synthesis domains whose data structures can be known beforehand. It is thus less easy to select suitable induction rules for precompilation. However, it is possible to apply particular well-founded relations to the domains, and it is also possible to infer the correct use of particular well-founded relations for a given domain arrangement.

Well-Founded Induction

A well-founded relation over the set $\mathcal{A}$ is a relation with no infinitely decreasing sequences of the form $(... \prec a_3 \prec a_2 \prec a_1)$ such that $a_i \in \mathcal{A}$ for all i. (A well-founded relation need not be transitive.) A consequence of the property of well-foundedness is a necessary and sufficient condition for the existence of a minimal element.

The set of natural numbers under ordering $<$ is well-founded. By contrast, the set of non-negative real numbers under the ordering $<$ is not well-founded due to the existence of sequences such as $(... < \frac{1}{4} < \frac{1}{2} < 1)$.

Well-founded induction is an inference rule with the generality to capture a wide class of induction rules. For a domain $\mathcal{A}$, well-founded relations $\mathcal{W}$, and proposition Φ, it may be represented as an object language axiom as follows.

$$\frac{\exists W \in \mathcal{W}, \forall x \in \mathcal{A} \,.\, (\forall y \in \mathcal{A} \,.\, y \prec_W x \rightarrow \Phi(y)) \rightarrow \Phi(x)}{\rightarrow \forall x \in \mathcal{A} \,.\, \Phi(x)}$$

Well-founded relations are of interest because of the role they play in the well-founded induction principle: to prove a goal $\forall x \Phi(x)$, it is sufficient, assuming an arbitrary well-founded relation named r, to consider an arbitrary entity x and prove $\Phi(x)$ assuming the induction hypothesis:

$$\forall u \,.\, \left[\begin{array}{c} \textit{Rel}(r, u, x) \rightarrow \\ \Phi(u) \end{array} \right]$$

We may thus prove $\Phi(x)$, under the induction hypothesis that $\Phi(u)$ holds for all entities u that are less than x under the well-founded relation named r. (In the case in which r is the $<$ relation over the natural numbers, this is called *complete induction*.) The action theory will be developed accordingly in the steps below.

The most significant part of instantiating the induction schema involves the selection of the well-founded relation. This selection therefore becomes part of the search for a proof and it becomes necessary to represent the ordering relation in the object language.

To achieve this, a sort $\mathcal{R}$ for *well-founded relations* (variables $r, r_1, \ldots$), is added to PPT, and to represent well-founded relation $\prec_W$, we introduce a distinguished predicate $\textit{Rel}(r, x, y)$ for the proposition that x and y are related through the reified relation named r. The relation $\textit{Rel}(r, x, y)$ is linked to W by the following. For all $\forall x, y \in \mathcal{A}$, $W \in \mathcal{W}$, $\forall r \in \mathcal{R}$, such that r names the relation W, $\textit{Rel}(r, x, y)$ holds if and only if $(x \prec_W y)$ holds.

According to the well-foundedness condition above, a well-founded relation r is one that admits no infinite decreasing sequences. There are thus no infinite sequences of entities $(x1, x2, x3....)$ such that

$$\textit{Rel}(r, x2, x1), \textit{Rel}(r, x3, x2) \text{ and } \textit{Rel}(r, x4, x3) \ ...$$

To illustrate the effect in a finite blocks world, the relation $\textit{Rel}(\text{covers}, a, b)$ says that a is less than b under r. It represents that the relation named *covers*, holding whenever block a is on top of block b, is well-founded because in a given situation there are no infinite towers.

For a well-founded relation r, a plan f and a property described by the formula Γ taking the arguments of situation s and input expression u, we instantiate the induction schema to the application setting as follows.

$$\frac{\exists r \forall s_0 \forall a. \left[\begin{array}{c} \forall s \forall u. \left[\begin{array}{c} \textit{Rel}(r, \langle s, u \rangle, \langle s_0, a \rangle) \rightarrow \\ \Gamma(s, u, f(u)) \end{array} \right] \\ \rightarrow \Gamma(s_0, a, f(a)) \end{array} \right]}{\rightarrow \forall s_0 \forall a \,.\, \Gamma(s_0, a, f(a))} \quad \text{(P15)}$$

The entailment relation for the setting of recursive plans now includes induction and may be stated as follows: assuming Γ, find a suitable ordering relation $\prec_r$ such that ProPT $\wedge$ $P15 \models P14$. The plan may be extracted from a constructive proof of the existence of z.

With the inclusion of induction however, the theory presents a difficulty in realization in an automated theorem proving environment. The induction schema expresses an infinite number of first order rule instances which must be available to take part in the proof as appropriate. To provide a solution to this problem, we will employ a technique based on skolemization to re-express induction in a form more suited to automated reasoning.

Plan formation occurs through proof of the existential plan term contained in the query. Since the proof of existence of a plan is constrained to be constructive (proof by contradiction is disallowed) it is possible to extract an entity r_0 representing a well-founded relation and an associated plan $f(a)$ that satisfies a condition $\Gamma(r_0, a, f(a))$ from a proof of a theorem $\exists r \forall a \exists z . \Gamma(r, a, z)$. In this goal, quantified variables are constrained to range over their respective sorts.

Further details and justification of the theoretical development contained in this section may be found in an extension to this paper at `http://www.ai.sri.com/~waldinge/recursiveplans`.

Inducing a Well-Founded Relation

The method of reasoning about workflows described in previous sections relies on the availability of suitable well-founded relations. The well-founded relation must match

the form of the inductive assertion needed for a proof of a particular goal.

We turn our attention now to the formation of specific well-founded relations from a set of abstract and general purpose orderings. The motivation for such an approach arises from the desire to make use of a small set of general well-founded relations rather than a particular well-founded relation targeted to each problem domain.

If we start with a well-founded relation r on entities of sort y, and a function f that maps entities of sort x into entities of sort y, we can induce on entities of sort x a well-founded relation *induce*(f, r), defined according to the following axiom P16.

$$\begin{array}{l} \mathit{Rel}(\mathit{induce}(f,r), x_1, x_2) \leftrightarrow \\ \quad \mathit{Rel}(r, f(x_1), f(x_2)) \end{array} \tag{P16}$$

As an example, let us consider taking x to be the sort of finite sets, y to be the sort of natural numbers, and f to be the cardinality function *card* that yields the number of elements in a given set. Then the relation *induce*$(\mathit{card}, <)$ is the relation that holds between two sets x_1 and x_2 if x_1 has fewer elements than x_2.

If r is well-founded over y then *induce*(f, r) is well-founded over x, because if $x_1, x_2, ...$ were an infinite decreasing sequence with respect to *induce*(f, r), then $f(x_1), f(x_2), ...$ would be an infinite decreasing sequence with respect to r, which is not possible.

Library Book Example Solution

We will now illustrate an application of this technique using the library example introduced earlier. Applying the recursion-introduction rule and skolemization from previous sections, we obtain the goal P17.

$$\exists r \exists z.(\hat{s_0}(r,z) \lhd z) \blacktriangleright \mathit{empty\text{-}set}(\mathit{book\text{-}set}_0(r)) \tag{P17}$$

We also may assume the induction hypothesis P18 during an attempt to find a well-founded relation r and a plan P19.

$$\forall r \forall s \forall u \,. \left[\begin{array}{c} \mathit{Rel}(r, \langle s, u\rangle, \langle \hat{s_0}(r,z), \mathit{book\text{-}set}_0(r)\rangle) \\ \rightarrow [(s \lhd \mathit{return\text{-}all}(\mathit{book\text{-}set}_0(r)) \blacktriangleright \\ \quad \mathit{empty\text{-}set}(\mathit{book\text{-}set}_0(r))] \end{array} \right] \tag{P18}$$

$$\mathit{return\text{-}all}(\mathit{book\text{-}set}_0(\mathrm{r})) \Leftarrow z \tag{P19}$$

P17 will achieve the specified goal condition from an arbitrary initial situation-input pair $\langle \hat{s_0}(r,z), \mathit{book\text{-}set}_0(r)\rangle$, under the inductive assumption stating that recursive calls to the desired plan will successfully achieve the specified condition for any situation-input pair $\langle s, u\rangle$ that is less than the initial situation-input pair.

The proof establishes the existence of the well-founded relation r and the associated plan z. During the proof, the well-founded relation r is taken to be the induced relation *induce*$(\rhd, r)$, defined by the axiom P20 where f is a function variable.

$$\begin{array}{l} \mathit{wfr}(\mathit{induce}(f, \mathrm{r}_1), \langle s_1, u_1\rangle, \langle s_2, u_2\rangle) \leftrightarrow \\ \quad \mathit{wfr}(r_1, f(s_1, u_1), f(s_2, r_2)) \end{array} \tag{P20}$$

The relation r_1 in turn is taken to be *proper-subset*. By our prior discussion of induced relations, we know that, since *proper-subset* is well-founded on finite sets, $\rhd$*proper-subset* is well-founded on pairs of situations and finite sets. The axiom tells us that, if the set *book-set*$_0$ is nonempty, the event *return-one*(*book-set*$_0$) yields a situation in which *book-set*$_0$ is a proper subset of its former self. In general, P21 holds.

$$\begin{array}{l} \mathit{if}\, s \blacktriangleright \mathit{not}(\mathit{empty\text{-}set}(\mathit{book\text{-}set})) \rightarrow \\ \mathit{wfr}(\mathit{proper\text{-}subset}, \\ \quad (s \lhd \mathit{return\text{-}one}(\mathit{book\text{-}set}) \rhd \mathit{book\text{-}set}) \\ \quad (s \rhd \mathit{book\text{-}set})). \end{array} \tag{P21}$$

Thus, in the case in which *book-set*$_0$ is non-empty, we have (by definition of the induced relation) P22. For compactness, *induce*(f, r) will be written as $(f\, r)$.

$$\begin{array}{l} \mathit{wfr}(\rhd \mathit{proper\text{-}subset}, \\ \quad \langle s \lhd \mathit{return\text{-}one}(\mathit{book\text{-}set}_0), \mathit{book\text{-}set}_0\rangle \\ \quad \langle s, \mathit{book\text{-}set}_0\rangle). \end{array} \tag{P22}$$

In the situation in which the book set is reduced, the goal of emptying the set entirely may be achieved by executing a recursive call. Taking s to be the initial situation $\hat{s_0}(r,z)$, and then taking r to set the well-founded relation $\rhd$ *proper-subset*, we can thus apply the induction hypothesis to conclude P23.

$$\begin{array}{l} (\hat{s_0}(\rhd \mathit{proper\text{-}subset}, z) \lhd \\ \mathit{return\text{-}one}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})) \lhd \\ \mathit{return\text{-}all}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})) \blacktriangleright \\ \mathit{empty\text{-}set}(\mathit{book\text{-}set}_0)) \end{array} \tag{P23}$$

For the case in which the set *book-set*$_0$ is already empty in the initial situation, we can achieve the desired goal by doing nothing, P24.

$$\begin{array}{l} (\hat{s_0}(\rhd \mathit{proper\text{-}subset}, z) \lhd \mathit{no\text{-}op}, \\ \quad \mathit{empty\text{-}set}(\mathit{book\text{-}set}_0)) \end{array} \tag{P24}$$

Using an application of the conditional introduction rules, we may now conclude that, in either case, the desired goal is achieved through a conditional term.

$$\begin{array}{l} (\hat{s_0}(\rhd \mathit{proper\text{-}subset}, z) \lhd \\ (\mathit{if}(\mathit{empty\text{-}set}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})), \\ \quad \mathit{no\text{-}op}, \\ \quad \mathit{return\text{-}one}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})) \lhd \\ \quad \mathit{return\text{-}all}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})) \blacktriangleright \\ \quad \mathit{empty\text{-}set}(\mathit{book\text{-}set}_0(\rhd \mathit{proper\text{-}subset})))). \end{array} \tag{P25}$$

Hence the original goal P1 is satisfied by the program P26.

$$\textit{return-all}(\textit{book-set}_0) \Leftarrow \textit{if empty-set}(\textit{book-set}_0), \quad \text{(P26)}$$
$$\textit{no-op},$$
$$\textit{return-one}(\textit{book-set}_0)\,;$$
$$\textit{return-all}(\textit{book-set}_0)$$

This section has provided an example demonstrating how an abstract well-founded relation $\rhd$*proper-subset* can be applied automatically to reasoning about a practical workflow whose purpose of to achieve a goal by a particular type of repetition, represented here as recursion.

Axiom Set for ATP

The theory presented above has been run on the theorem prover SNARK (Stickel, Waldinger, and Chaudhri 2001) using the axioms presented in this section. The plan already shown in P26 was generated automatically as output. In these axioms, variables are prefixed by a question mark for greater clarity. Further details may be found in the extension to this paper referred to previously.

The goal must serve two purposes and accordingly is chosen to be a conjunction of the following atoms: $\textit{pair}(?e, ?\textit{wfr})$ and $holds(result(s_0(?\textit{wfr}, ?e), ?e), empty\text{-}setx(\textit{book-setx}_0 ?\textit{wfr}))$. The first atom serves to identify the two answer variables, and the second forms the goal condition.

$$eq(?x, ?x) \quad \text{(P27)}$$
(Reflexivity of equality.)

$$\textit{result}(?s, \textit{no-op}) = ?s \quad \text{(P28)}$$
(Result *no-op* no op.)

$$\textit{result}(\textit{result}(?s, ?e_1), ?e_2) = \quad \text{(P29)}$$
$$\textit{result}(?s, \textit{compose}(?e_1, ?e_2))$$
(Result of composition of events.)

$$\textit{compose}(\textit{no-op}, ?e) = ?e \quad \text{(P30)}$$
(Compose *no-op* and event is event.)

$$\textit{compose}(?e, \textit{no-op}) = ?e \quad \text{(P31)}$$
(Compose event and *no-op* is event.)

$$\textit{Holds}(?s, not(?p)) \leftrightarrow \neg \textit{Holds}(?s, ?p) \quad \text{(P32)}$$
(Transparency of not.)

$$\textit{Holds}(\textit{result}(?s, \textit{if}(?p_0, ?e_1, ?e_2)), ?p) \leftarrow \quad \text{(P33)}$$
$$[\textit{Holds}(?s, ?p_0) \wedge \textit{Holds}(\textit{result}(?s, ?e_1), ?p))]$$
(If introduction positive.)

$$\textit{Holds}(\textit{result}(?s, \textit{if}(?p_0, ?e_1, ?e_2)), ?p) \leftarrow \quad \text{(P34)}$$
$$[\neg \textit{Holds}(?s, ?p_0) \wedge \textit{Holds}(\textit{result}(?s, ?e_2), ?p))]$$
(If introduction negative.)

$$\textit{Holds}(\textit{result}(?s, \textit{return-all}(?\textit{book-setx})) \quad \text{(P35)}$$
$$(\textit{empty-setx}(\textit{book-setx}_0, ?\textit{wfr}))) \leftarrow$$
$$\textit{and}(\textit{Rel}(?\textit{wfr}, \textit{pair}?s, ?\textit{book-setx}),$$
$$\textit{pair}(s_0(?\textit{wfr}, ?e), \textit{book-setx}_0(?wfr))))$$
(Recursive call achieves empty setx in reduced situation.)

$$\textit{Rel}(\textit{induce}(?\textit{fun}, ?\textit{wfr}), \quad \text{(P36)}$$
$$\textit{pair}(?s_1, ?ex_1), \textit{pair}(?s_2, ?ex_2)) \leftarrow$$
$$\textit{Rel}(?\textit{wfr}, \textit{apply}(?\textit{fun}, ?s_1, ?ex_1)$$
$$\textit{apply}(?\textit{fun}, ?s_2, ?ex_2)))$$
(Definition of induced by fun.)

$$\textit{apply}(\textit{val-setx}, ?s, ?\textit{book-setx}) = \quad \text{(P37)}$$
$$\textit{val-setx}(?s, ?\textit{book-setx})$$
(Apply val setx linkage.)

$$\textit{Rel}(\textit{proper-subset}, \textit{val-setx}(result(?s, \quad \text{(P38)}$$
$$\textit{return-one}(?\textit{book-setx})), ?\textit{book-setx})$$
$$\textit{val-setx}(?s, ?\textit{book-setx})) \leftarrow$$
$$and(\textit{Holds}(?s, not(\textit{empty-setx}(?\textit{book-setx}))))$$
(Return one makes book set smaller.)

Satisfaction of Workflow Control Patterns

The representational power of the conditional recursive plans defined in the previous sections is sufficient to describe many important workflow control constructs. (Aalst et al. 2003) identified distinct patterns of control that are typically required in workflow expression. In this section, we have selected four particular patterns for their foundational characteristics, and we identify the corresponding PPT constructs that realize them.

Workflow pattern: *Sequential Activities*. The Sequence control pattern provides for a sequential execution of a set of predefined activities, such that each activity begins only after its predecessor has completed. This is represented in PPT as a composition of events P39.

$$\textit{routine-A}(\rho) \Leftarrow \textit{activity}_1(\rho), \quad \text{(P39)}$$
$$\textit{activity}_2(\rho),$$
$$\textit{activity}_n(\rho).$$

Workflow pattern: *Exclusive Choice* (conditional). The Exclusive Choice (case statement) control pattern branches to one of a number of new paths, subject to conditions associated with each path. However, only one path is ever chosen even though multiple conditions may be satisfied. The key to exclusive choice is to create a priority in the case analysis. This ensures we prevent more than one new path being started in the event that more than one condition is true. P40 describes exclusive choice.

$$\textit{routine-A}(\rho) \Leftarrow \textit{if condition}_1(\rho), \quad \text{(P40)}$$
$$\textit{body-activities}_1(\rho),$$
$$\textit{if condition}_2(\rho),$$
$$\textit{body-activities}_2(\rho),$$
$$\textit{if condition}_n(\rho),$$
$$\textit{body-activities}_n(\rho).$$

Workflow pattern: *Iteration* (while). The iteration pattern supports the execution of an activity or process multiple times. The structured loop representation provided for in

this formalism is the *while loop*. This uses a pre-test condition evaluated at the beginning of the loop prior to a conditioned execution entry. Iteration is supported here by tail recursion, and takes the form of P41.

$$\begin{aligned} \textit{routine-A}(\rho) \Leftarrow\ & \textit{if condition}(\rho), \\ & [\textit{body-activity}(\rho); \\ & \textit{routine-A}(\delta(\rho))], \\ & \textit{no-op}. \end{aligned} \tag{P41}$$

Workflow pattern: *Recursion*. Recursive structures defines an activity in terms of itself. Although the inclusion of recursion over iteration does not increase theoretical expressivity, it still offers the chance to represent certain process structures more efficiently than can be achieved using iteration. It also provides greater expressivity for the acceptance of user-generated workflows for reasoning.

A *construct* consists of either
- an *activity*(ρ), or
- a *conditional if*(ρ, *construct*$_1$, *construct*$_2$), or
- a recursive call *routine-A*($\delta(\rho)$).

Then, a recursive workflow consists of the following:

$$\begin{aligned} \textit{routine-A}(\rho) \Leftarrow\ & \textit{construct}_1(\rho), \\ & \textit{construct}_2(\rho), \\ & \textit{construct}_n(\rho). \end{aligned} \tag{P42}$$

The property of termination of recursive structures applies in a similar way to iteration in that we do not wish to have an infinite number of recursive decompositions. This property is linked to the existence of well-foundedness in the corresponding induction rule used for recursion introduction. If the particular rule is based on a well-founded ordering, then we are guaranteed that the recursion will terminate.

These four control patterns together provide workflow expressivity that is Turing complete. The four control patterns are therefore (technically) capable of expressing any workflow. (Strictly speaking, the presence of either pattern 3 or 4 along with 1 and 2 is sufficient for Turing completeness.) The absence of constructs for the representation of concurrent processes is one dimension of enhancement necessary for a more full workflow representation.

Specification and Consequences of Workflows

Specifying the goals that a process is to archive is an accepted method of process definition. However, there is frequently information available about a process that takes alternative forms. As well as specifying the goals of a desired workflow using P14, we can additionally state a requirement that specific actions be incorporated into the workflow activities. For instance, consider a desire to incorporate the use of a taxi into a plan of travel. This may be achieved by substituting the plan variable z in P14 for $e_1 \circ taxi() \circ e_3$.

Through the use of the two complementary methods of goal enumeration, and action enumeration, it is possible to specify both *what* a task is to achieve and *how* it is to function. Such a dual representation is able to offer flexibility and power for the specification of workflows in a practical environment.

Evaluation and Discussion

There are disparate prior approaches to reasoning about recursive structures. Systems for modeling workflows such as (Kim, Spraragen, and Gil 2004) can typically represent recursive process structures, but not reason about their properties. This remains the case even when workflow modeling is given a logical account, as in (Cicekli and Cicekli 2006).

Planning solutions may be applied to workflow creation. However, it has been difficult to reason about recursive plan structures. Software synthesis methods (e.g. structural induction in (Boyer, R. S. and Moore, J S. 1979)) may be applied by viewing the workflow as a program. However, some synthesis methods are not immediately applicable to constructing processes from goals. Methods based on program transformation (Burstall and Darlington 1977) rely on a specification to contribute some structure. When the specification takes the form of goals, there is no original program structure to transform.

As a consequence, (Boyer, R. S. and Moore, J S. 1979) does not prove goals with existential quantification, as required for plan synthesis, and it also makes use of the specification's recursive structure in Recursion Analysis, the means for induction rule selection.

(Levesque 2005) augmented planning techniques with inductive generalization as an alternative means. However, the generated plans are not guaranteed to be correct. (Magnusson and Doherty 2008) used fixed-point induction to reason about iterative structures. However, synthesized plans are only partially correct, and this allows the creation of non-terminating plans.

In this work, we combine techniques for deductive planning and software synthesis to reason about and extract recursive plans. (See (Constable and Moczydlowski 2006) for alternative methods of extraction.) The method is based on well-founded induction, and therefore all synthesized plans are guaranteed to be correct and to terminate. (Bundy et al. 2006) uses several specially developed induction rules for this purpose; however we use a single induction schema instantiated with a well-founded relation. (Ball et al. 2006; Hurd 2007) also dynamically construct a well-founded relation, but this is in the software verification setting only.

(Boyer, R. S. and Moore, J S. 1979) provide for automatic generalization of the inductive hypothesis. However the generalizations required in practice are frequently not discovered by the theorem prover.

Conclusion

We have developed a process language and theory for workflows that when combined with the method of well-founded induction demonstrates the creation of recursive process workflows which are guaranteed to be correct and to terminate. Several examples have been demonstrated to work on a theorem prover. A single induction schema is used, and specialization of inductive inference is achieved with the selection of a well-founded relation. The method employs a small number of abstract well-founded relations, which are then used as a basis to induce specific relations required for the reasoning.

Since reasoning of this type is undecidable, future work will examine, in more depth, suitable heuristics to accompany the approach. The method of inductive hypothesis generalization also promises to further increase the class of problems for which the techniques described can be applied. This will also be the subject of further research.

Acknowledgements

We would like to thank Sheila McIlraith for insightful discussions. This work was supported in part by a grant from the National Science Foundation.

References

[Aalst et al. 2003] Aalst, W.; Hofstede, A.; Kiepuszewski, B.; and Barros, A. P. 2003. Workflow patterns. *Distributed and Parallel Databases* 14(1):5–51.

[Baker 1989] Baker, A. B. 1989. A simple solution to the yale shooting problem. In Brachman, H. J. L. R. J., and Reiter, R., eds., *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, 11–20. Toronto, Canada: Morgan Kaufmann.

[Ball et al. 2006] Ball, T.; Bounimova, E.; Cook, B.; Levin, V.; Lichtenberg, J.; McGarvey, C.; Ondrusek, B.; Rajamani, S. K.; and Ustuner, A. 2006. Thorough static analysis of device drivers. In *EuroSys'06*, 73–85. ACM.

[Boyer, R. S. and Moore, J S. 1979] Boyer, R. S., and Moore, J S. 1979. *A Computational Logic*. New York: Academic Press.

[Bundy et al. 2006] Bundy, A.; Dixon, L.; Gow, J.; and Fleuriot, J. D. 2006. Constructing induction rules for deductive synthesis proofs. *Electr. Notes Theor. Comput. Sci* 153(1):3–21.

[Burstall and Darlington 1977] Burstall, R. M., and Darlington, J. 1977. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery* 24(1):44–67.

[Burstall 1969] Burstall, R. M. 1969. Proving properties of programs by structural induction. *The Computer Journal* 12(1):41–48.

[Cicekli and Cicekli 2006] Cicekli, N. K., and Cicekli, I. 2006. Formalizing the specification and execution of workflows using the event calculus. *Inf. Sci* 176(15):2227–2267.

[Clark and van Emden 1981] Clark, K. L., and van Emden, M. H. 1981. Consequence verification of flowcharts. *IEEE Transactions on Software Engineering* Se-7(1):52–60.

[Constable and Moczydlowski 2006] Constable, R. L., and Moczydlowski, W. 2006. Extracting programs from constructive HOL proofs via IZF set-theoretic semantics. In Furbach, U., and Shankar, N., eds., *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, 162–176. Springer.

[Floyd 1967] Floyd, R. W. 1967. Assigning meanings to programs. *Proceedings of the Symposium on Applied Math, American Mathematical Society* XIX:19–32.

[Hurd 2007] Hurd, J. 2007. Proof pearl: The termination analysis of terminator. In Schneider, K., and Brandt, J., eds., *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*, volume 4732 of *Lecture Notes in Computer Science*, 151–156. Springer.

[Kim, Spraragen, and Gil 2004] Kim, J.; Spraragen, M.; and Gil, Y. 2004. An intelligent assistant for interactive workflow composition. In Vanderdonckt, J.; Nunes, N. J.; and Rich, C., eds., *Intelligent User Interfaces*, 125–131. ACM.

[Kraan, Basin, and Bundy 1993] Kraan, I.; Basin, D.; and Bundy, A. 1993. Middle-out reasoning for logic program synthesis. In Warren, D. S., ed., *Proceedings of the Tenth International Conference on Logic Programming*, 441–455. Budapest, Hungary: The MIT Press.

[Levesque 2005] Levesque, H. J. 2005. Planning with loops. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, 509–515. Professional Book Center.

[Magnusson and Doherty 2008] Magnusson, M., and Doherty, P. 2008. Deductive planning with inductive loops. In Brewka, G., and Lang, J., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, 528–534. AAAI Press.

[Manna and Waldinger 1987] Manna, Z., and Waldinger, R. 1987. How to clear a block: a theory of plans. *Journal of Automated Reasoning* 3:343–377.

[McCarthy and Hayes 1969] McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Chichester, England: Edinburgh University Press. 463–502.

[Park 1970] Park, D. 1970. Fixpoint induction and proof of program semantics. In Melzer, B., and Michie, D., eds., *Machine Intelligence*, volume 5. Edinburgh University Press. 59–78.

[Reiter 1991] Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation*. Cambridge, Massachusetts.: Academic Press. 359–380.

[Shanahan 1997] Shanahan, M. 1997. *Solving the Frame Problem: A Methematical Investigation of the Common Sence Law of Inertia*. Artificial Intelligence. Cambridge, Massachusetts, USA: The MIT Press.

[Stickel, Waldinger, and Chaudhri 2001] Stickel, M. E.; Waldinger, R. J.; and Chaudhri, V. K. 2001. A guide to SNARK. SRI International. http:www.ai.sri.com/snark/tutorial/tutorial.html.