Incremental Parametric Development of Greedy Algorithms

Dominique Cansell, Dominique Méry

Incremental Parametric Development of Greedy Algorithms

Dominique Cansell[1] ,[2]

LORIA
Université de Metz
Ile du Saulcy
57045 Metz, France

Dominique Méry[1] ,[3]

LORIA
Université Henri Poincaré Nancy 1
BP 239
54506 Vandoeuvre-lès-Nancy, France

Abstract

The event B method provides a general framework for modelling both data structures and algorithms. B models are validated by discharging proof obligations ensuring safety properties. We address the problem of development of greedy algorithms using the seminal work of S. Curtis; she has formalised greedy algorithms in a relational calculus and has provided a list of results ensuring optimality results. Our first contribution is a re-modelling of Curtis's results in the event B framework and a mechanical checking of theorems on greedy algorithms The second contribution is the reuse of the mathematical framework for developing greedy algorithms from event B models; since the resulting event B models are generic, we show how to instantiate generic event B models to derive specific greedy algorithms; generic event B developments help in managing proofs complexity. Consequently, we contribute to the design of a library of proof-based developed algorithms.

Keywords: Formal method, B event-based method, refinement, safety, greedy algorithms.

1 Introduction

Algorithms provide a class of systems on which one can apply proof-based development techniques like the event B method, especially the refinement. The main advantage is the fact that we teach data structures and algorithms to students, who should have simple explanations of why a given algorithm is effectively working or

[1] We thank anonymous referees for their comments; we thank Emilie Balland who has developed preliminary models of the study, when she was in the MOSEL team for a training period under the supervision of Dominique Méry. J.-R. Abrial supports our works by friendly messages and comments. Sharon Curtis provides us comments on early developed models.

[2] Email: cansell@loria.fr

[3] Email: mery@loria.fr

A revised version of this paper will be electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

why some assertion is an invariant for the algorithm under consideration ... Hence, we have a good knowledge of algorithmic problems and it is simpler for us to apply proof-based development techniques on the algorithmic problems. Greedy algorithms constitute a well defined class of algorithms (applications and properties) and we aim to provide proof-based patterns for facilitating the proof-based development (in B) of greedy algorithms.

In a previous work [4], we have developed Prim's algorithm and we have proved properties over trees: the inductive definition of trees helps in deriving intermediate lemmas asserting that the growing tree converges to the minimal spanning tree, according to the greedy strategy. The resulting algorithm was completely proved using the proof assistant [7] and we can partially reuse current developed models to obtain Dijkstra's algorithm or Kruskal's algorithm. The greedy strategy is not always optimal and the optimality of the resulting algorithm is proved by the theorem 24.1 of Cormen's book [8] in the case of the minimal spanning tree problem. The gain is clear, since we had a mechanised and verified proof of Prim's algorithm. The formalisation of greedy-oriented algorithmic structures was not so complicated but we were assuming that a general theory on greedy structures could help in designing our greedy algorithms using the event B method. Fortunately, S. Curtis [9] brings the theoretical material that was missing in our project; she has formalised in a relational framework properties required for leading to the optimality of solutions, when applying a greedy technique. However, we have not explained why we are choosing the greedy method and what for? Our quest is to propose general proof-based developments (or patterns) for a given problem or for a given paradigm. We think that the refinement provides a way to introduce generic elements in developed models. A second objective is to illustrate the adequacy of the B prover [7], when checking results over set-theoretical structures; in a sense, our work may seem to be a plagiarism of Curtis's paper, but the tool scans each detail to check and it validates each user hint, and, generally, there is no assisted significant proof without human hint (proof step or tricky lemma). Hence, our paper is an exercise in checking properties over greedy structures and in proposing generic development of greedy algorithms; we do not know any other mechanized complete proof-based developments of greedy algorithms.

1.1 Greedy algorithms

Greedy algorithms are used to solve optimization problems like the shortest path problem or the best order to execute a set of jobs. A greedy algorithm works in a local step to satisfy a global constraint. A greedy algorithm can be summarized by the general algorithm 1, where C is the set of candidates and S is the set containing the solution or possibly no solution. The goal is to optimize a set of candidates which is a solution to the problem; the optimization maximizes or minimizes the value of an objective function. The optimization state is checked by the Boolean function called *goodchoice*. Lectures notes of Charlier [6] provide a very complete introduction to the underlying theories of the greedy algorithms like the matroids theory for instance. S. A. Curtis [9] classifies greedy algorithms and uses a relation framework for expressing properties of the greedy algorithms; her characterization is based on the preservation of the safety properties but the termination part is

Pre-condition:*C is a set of possible candidates*
Post-condition:*Either a solution S, or no solution does exist*

```
BEGIN
  S:= emptyset;
  WHILE C # emptyset and not solution(S)
    DO
      x:=select(C); C:=C-{x};
      IF goodchoice(S \/ { x }) THEN S:=S \/ { x } FI;
    OD;
  IF solution(S) THEN return S ELSE return(no_solution} FI;
END
```

Algorithm 1. General Greedy Algorithm [8]

missing. Our paper is based on her works, it reformulates properties and proposes mechanically checked proofs in the B prover engine [3,7]. First, we translate the mathematical notations of the models of Curtis; we check the results proved in the paper using the theorem prover of B. Then we show how to develop a greedy algorithm according to a given assumption. Finally, we show how to instantiate a specific problem that can be solved using the greedy strategy.

1.2 Summary of the paper

Section 2 provides an introduction to the event B method and a definition of instantiation mechanisms that can be used to introduce generic developments. Section 3 details the mathematical structures underlying the greedy method and reformulates in event B the notations due to Curtis. Curtis's proofs are completely checked with the proof tool. Section 4 develops a very general greedy algorithmic and shows how mathematical properties are introduced while developing. An abstract greedy algorithm is derived from the last refinement model of the development and an instantiation is given. Section 5 concludes the work with possible perspectives.

2 On the event B development

2.1 Event-based modelling

Our event-driven approach [2] is based on the B notation [1]. It extends the methodological scope of basic concepts such as set-theoretical notations and generalized substitutions in order to take into account the idea of *formal models*. Roughly speaking, a formal model is characterized by a (finite) list x of *state variables* possibly modified by a (finite) list of *events*; an invariant $I(x)$ states some properties that must always be satisfied by the variables x and *maintained* by the activation of the events. In what follows, we briefly recall definitions and principles of formal models and explain how they can be managed by the tool [3,7].

Definition 2.1 Generalized substitutions are borrowed from the B notation. They provide a way to express the transformations of the values of the state variables of a formal model. In its simple form, $x := E(x)$, a generalized substitution looks like an assignment statement. In this construct, x denotes a vector build on the set of state variables of the model, and $E(x)$ a vector of expressions of the same size as the vector x. The interpretation we shall give here to this statement is *not* however that of an assignment statement. We interpret it as a *logical simultaneous substitution* of each variable of the vector x by the corresponding expression of the

vector $E(x)$. There exists a more general form of generalized substitution. It is denoted by the construct $x : P(x_0, x)$. This is to be read: "x is modified in such a way that the predicate $P(x_0, x)$ holds", where x denotes the *new value* of the vector, whereas x_0 denotes its *old value*. It is clearly non-deterministic in general. This general form could be considered as a *normal form*, since the simplest form $x := E(x)$ is equivalent to the more general form $x : (x = E(x_0))$.

Definition 2.2 An event is essentially made of two parts: a *guard*, which is a predicate built on the state variables, and an *action*, which is a generalized substitution. An event can take one of the forms shown in the table below. In these constructs, *evt* is an identifier: this is the event name. The first event is not guarded: it is thus always enabled. The guard of the other events, which states the necessary condition for these events to occur, is represented by $G(x)$ in the second case, and by $\exists\, t \cdot G(t, x)$ in the third one. The latter defines a non-deterministic event where t represents a vector of distinct local variables. The, so-called, before-after predicate $BA(x, x')$ associated with each event shape, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before (x) and just after (x') the event "execution".

Event	Before-after Predicate $BA(x, x')$
$evt \;\widehat{=}\;$ BEGIN $x : P(x_0, x)$ END	$P(x, x')$
$evt \;\widehat{=}\;$ WHEN $G(x)$ THEN $x : Q(x_0, x)$ END	$G(x) \;\wedge\; Q(x, x')$
$evt \;\widehat{=}\;$ ANY t WHERE $G(t, x)$ THEN $x : R(x_0, x, t)$ END	$\exists\, t \cdot (\, G(t, x) \;\wedge\; R(x, x', t)\,)$

The generalized substitution $x :\in S(x)$ is equivalent to the normal form $x : (x \in S(x0))$ and the event $evt \;\widehat{=}\;$ BEGIN $x :\in S(x)$ END is equivalently written as follows: $evt \;\widehat{=}\;$ BEGIN $x : (x \in S(x0))$ END. Proof obligations are produced from events in order to state that the invariant condition $I(x)$ is preserved. We next give the general rule to be proved. It follows immediately from the very definition of the before-after predicate, $BA(x, x')$ of each event:

$$I(x) \;\wedge\; BA(x, x') \;\Rightarrow\; I(x')$$

Notice that it follows from the two guarded forms of the events that this obligation is trivially discharged when the guard of the event is false. When it is the case, the event is said to be "disabled".

2.2 Model Refinement

The refinement of a formal model allows us to enrich a model in a *step by step* approach. Refinement provides a way to construct stronger invariants and also to add details in a model. It is also used to transform an abstract model in a more concrete version by modifying the state description. This is essentially done by

extending the list of state variables (possibly suppressing some of them), by refining each abstract event into a corresponding concrete version, and by adding new events. The abstract state variables, x, and the concrete ones, y, are linked together by means of a, so-called, *gluing invariant* $J(x, y)$. A number of proof obligations ensure that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines *skip*, (3) no new event take control for ever, and (4) relative deadlock-freeness is preserved.

Definition 2.3 We suppose that an abstract model AM with variables x and invariant $I(x)$ is refined by a concrete model CM with variables y and gluing invariant $J(x, y)$. If $BAA(x, x')$ and $BAC(y, y')$ are respectively the abstract and concrete before-after predicates of the same event, we have to prove the following statement:

$$\boxed{I(x) \ \wedge \ J(x,y) \ \wedge \ BAC(y,y') \ \Rightarrow \ \exists x' \cdot (BAA(x,x') \ \wedge \ J(x',y'))}$$

This says that under the abstract invariant $I(x)$ and the concrete one $J(x, y)$, a concrete step $BAC(y, y')$ can be simulated ($\exists x'$) by an abstract one $BAA(x, x')$ in such a way that the gluing invariant $J(x', y')$ is preserved. A new event with before-after predicate $BA(y, y')$ must refine *skip* ($x' = x$). This leads to the following statement to prove:

$$\boxed{I(x) \ \wedge \ J(x,y) \ \wedge \ BA(y,y') \ \Rightarrow \ J(x,y')}$$

Moreover, we must prove that a variant $V(y)$ is decreased by each new event (this is to guarantee that an abstract step may occur). We have thus to prove the following for each new event with before-after predicate $BA(y, y')$:

$$\boxed{I(x) \ \wedge \ J(x,y) \ \wedge \ BA(y,y') \ \Rightarrow \ V(y') < V(y)}$$

Finally, we must prove that the concrete model does not introduce more deadlocks than the abstract one. This is formalized by means of the following proof obligation:

$$\boxed{I(x) \ \wedge \ J(x,y) \ \wedge \ \mathsf{grds}(AM) \ \Rightarrow \ \mathsf{grds}(CM)}$$

where $\mathsf{grds}(AM)$ stands for the disjunction of the guards of the events of the abstract model, and $\mathsf{grds}(CM)$ stands for the disjunction of the guards of the events of the concrete one.

The refinement of models by refining events is close to the refinement of action systems, the refinement of UNITY and the TLA refinement, even if there is no explicit semantics based on traces but one can consider the refinement of events like a relation between abstract traces and concrete traces. The stuttering plays a central role in the global process of development where new events can be added into the refinement model. When one refines a model, one can either refine an existing event by strengthening the guard or/and the before-after predicate (removing non-determinism), or add a new event which is supposed to refine the skip event. When one refines a model by another one, it means that the set of traces of the refined model contains the traces of the resulting model with respect to the stuttering relationship. Models and refined models are defined and can be validated through the

proofs of proof obligations; the refinement supports the proof-based development.

We summarize set-theoretical notations that can be used in the writing of formal definitions related to constants. In fact, the modelling of data is oriented by sets, relations and functions.

Name	Syntax	Definition
Binary Relation	$s \leftrightarrow t$	$\mathcal{P}(s \times t)$
Composition of relations	$r; s$	$\{x, y \mid x \in \mathcal{P}(a) \quad \wedge \quad y \in \mathcal{P}(b) \quad \wedge \quad \exists z.(z \in \mathcal{P}(c) \quad \wedge \quad x, z \in r \quad \wedge \quad z, y \in s)\}$
Inverse relation	r^{-1}	$\{x, y \mid x \in \mathcal{P}(a) \quad \wedge \quad y \in \mathcal{P}(b) \quad \wedge \quad y, x \in r\}$
Domain	$\mathsf{dom}(r)$	$\{a \mid a \in s \quad \wedge \quad \exists b.(b \in t \quad \wedge \quad a \mapsto b \in r)\}$
Range	$\mathsf{ran}(r)$	$\mathsf{dom}(r^{-1})$
Identity	$\mathsf{id}(s)$	$\{x, y \mid x \in s \quad \wedge \quad y \in s \quad \wedge \quad x = y\}$
Image	$r[w]$	$\mathsf{ran}(\mathsf{id}(w); r)$
Partial Function	$s \nrightarrow t$	$\{r \mid r \in s \leftrightarrow t \quad \wedge \quad (r^{-1}; r) \subseteq \mathsf{id}(t)\}$
Total Function	$s \longrightarrow t$	$\{f \mid f \in s \nrightarrow t \quad \wedge \quad \mathsf{dom}(f) = s\}$

2.3 Parametric development in the B event-based method

The B method provides a framework for developing parametric models of systems; it means that a problem is defined by parameters to instantiate and that a abstract B development already exists. When we use the expression *problem*, it means that we are able to relate the current problem to solve and the abstract problem solved by the already existing B development. The idea is to assume that the mathematical framework is given and that there are some constants which are remaining to set up. However, the main problem is to check that the effective parameters satisfy the constraints of the theory. We have not developed a parametrization mechanism for the B event-based method but we indicate how the current framework can be used for implementing the instantiation. In a draft paper [5], we have proposed a mechanism for instantiating B developments and we have used the searching problem [10]. First at all, we review what is a validated project in the B methodology. A validated project is a collection of models, either machine or refinement or implementation, which are checked by the different processes: type checking and proving. We assume that every model is completely proved by tools. The link between models is the refinement and it is already checked and completely proved. We assume that there is only one machine in the current project because we focus on the re-usability of developed models. Hence, a project $\mathbb{G}$ is roughly speaking an acyclic directed graph of models related by the refinement relationship: $\mathbb{G} = (\{G, ..., G_n\}, \longrightarrow)$.

The model G plays the role of the initial machine for the development and the project is simply defined by the model G. In order to avoid confusion among names, we use different fonts for designating problems, models and projects. The creation and the development of the project $\mathbb{G}$ follow the event B methodology. We assume that $\mathbb{G}$ is an existing project corresponding to a given *generic* problem to solve. The problem to solve is denoted $\mathcal{G}$ and the project $\mathbb{G}$ is supposed to solve the given problem. The model G is, in fact, the statement of the generic problem and clauses of the project $\mathbb{G}$ recall the parameters of the problem and of the model G.

MODEL

G

SETS

s

CONSTANTS

c

PROPERTIES

$P(s, c)$

VARIABLES

x

INVARIANT

$J(x)$

ASSERTIONS

$A(x)$

INITIALISATION

$S(x)$

EVENTS

$event = L(x)$

END

The model G provides a general framework for the current problem; the problem is characterized by a theory defined by the clauses SETS, CONSTANTS and PROPERTIES. The unique event helps in solving the problem, which is in fact defined with respect to the model itself. The event *event* defines the problem like a pre/post-condition between the initial state of y defined by the event *init* and the final state resulting from the event *event*. Among models of the project, models play a role of initialisation in the development process; we recall that we assume that there is only one model called G which roots a development following different paths in the graph of models. G contains clauses defining constants, properties, variables, invariant, assertions and operations. We summarize the discussion on project and problem.

> A project $\mathbb{G}$ is related to a problem $\mathcal{G}$ and it is developed from a model G stating the problem $\mathcal{G}$ and assumptions on the problem. It contains refined models of the model G.

Carrier sets and constants can be instantiated but proof obligations must be proved to ensure the validity of properties, which are theorems in an instantiation. The model G is the set of the development by instantiation.
The model G is the starting point of the development of the project $\mathbb{G}$ and it solves the problem $\mathcal{G}$; the project is formally checked by the theorem prover. The model G is supposed to solve a problem stated by the event *event* (the *greedy* problem solves the problem of optimisation, for instance): the problem is said to be a generic problem.

The instantiation of the generic project $\mathbb{G}$ corresponds to a given problem $\mathcal{P}$; we state the specific problem to solve, using a new model P. The new problem to solve involves sets t, constants d, properties over constants $A(t, d)$, variables y different from x, invariant over variables y namely $I(y)$ and finally *spec_event* states the specific problem. We do not focus on the process of writing the specific problem but it might be obtained after a separate step-by-step development; it appears to be relatively obvious to state. Let us write the new problem as a new system.

The instantiation of the generic project $\mathbb{G}$ for the generic problem $\mathcal{G}$ to solve the specific problem $\mathcal{P}$ consists of exhibiting a set term $\sigma(t, d)$, defined in terms of the set and constant of t, and also a similar constant term $\gamma(t, d)$ able to instantiate the constant c of G. The instantiation consists thus of repainting s and c in G with $\sigma(t, d)$ and $\gamma(t, d)$ and to invoke it as $G\,(\sigma(t, d), \gamma(t, d))$. We must also rename each variable (resp. event) of G by an unique variable (resp. event) of P. This instantiation must resolve the specific problem $\mathcal{P}$ and we propose to instantiate a development in a refinement of P. In fact, proof obligations of refinement assume

that the instantiated development solve the specific problem $\mathcal{P}$. We assume that P and G have no common parameters: x is different of y and events names are different. The next refinement states what does mean that the specific problem $\mathcal{P}$ is solved like the problem $\mathcal{G}$, after a suitable instantiation. When the instantiation is proved to be correct, we freely obtain a complete instantiated development for the new problem $\mathcal{P}$. We have to prove only a small part of instantiated proof obligations. An instantiation requires to prove that properties of the system G are theorems with respect to the properties of P:

MODEL

P

SETS

t

CONSTANTS

d

PROPERTIES

$A(s, c)$

VARIABLES

y

INVARIANT

$I(x)$

INITIALISATION

$spec_i nit(y)$

EVENTS

$spec_e vent = K(y)$

END

- The properties of the system G ie axioms defining the theory of G are theorems in the new theory defined by the problem P:

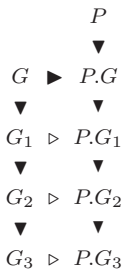
$$A(t, d) \Rightarrow P(\sigma(t, d), \gamma(t, d))$$

- both models are solving the same problem and the event *spec_event* of P is refined by the instance of the event *event* of G for the problem P; it is a classical instantiation:

$$\begin{array}{l} A(t, d) \wedge P(\sigma(t, d), \gamma(t, d)) \wedge I(y) \wedge \\ [s, c := \sigma(t, d), \gamma(t, d)]J(y1) \wedge y = y1 \wedge \\ BA(L)(y1, y1') \\ \Rightarrow \\ \exists y'.(I(y') \wedge BA(K)(y, y')) \end{array}$$

Property 1 *When the refinement is proved, the new problem $\mathcal{P}$ is solved by the development of the problem $\mathcal{G}$, up to renaming and instantiation.*

A new project $\mathbb{P}$ is created from the project $\mathbb{G}$ of the problem $\mathcal{G}$: events are renamed, variables are renamed, instantiations are done. Parameters are not completely instantiated or renamed; if a parameter is not instantiated, it keeps properties stated in the general model and no new proof obligation is generated. We summarize the links between models, problems and projects in the next diagram.



The diagram tells us where proof obligations must be proved: bold symbols express that new proof obligations are generated and one has to prove them; on the contrary, others symbols express that proof obligations have been generated but are inherited from a previous project. The diagram focuses on two directions for extension: a top-down extension and a left-right extension. We do not detail the mechanism of instantiation which is to implement in the B event-based approach and which was introduced in the paper [5].

3 Mathematical structures for the greedy method

First, in the step-by-step development, the definition of the mathematical objects requires to ensure the existence of a solution and to identify the problem to solve. The greedy method is effective, when properties are satisfied by the underlying mathematical structures. Like S. Curtis [9], we first define operators over relations and then we prove properties related to the greedy method. In fact, it will be a checking phase of Curtis's results: we use a more conventional way to write set-theoretical objects in the B notation.

3.1 Mathematical definitions

We assume that a set E is given and is not empty; we use operators over binary relations over E. Those operators are defined as constants and have properties: *rep*, *opt*, *quotient*, DOMAIN, NOTDOMAIN, *lambda*, *greedy*.

The relation *quotient* captures the idea of implication; in fact, it satisfies the property: $(quotient(S,T);T) \subseteq S$ which explains the choice of the name quotient. The domains of $(quotient(S,T)$ and S have the same carrier set. The range of $(quotient(S,T)$ and the domain of T have the same carrier set. The range of T and the range of S have the same carrier set.

If S is a relation between X and Y and T is a relation between Z and Y, then $quotient(S,T)$ is a relation between X and Z. The *set* operator does not exist in B but we can easily define it by quantification over domain and range. Hopefully, S. Curtis uses only two kinds of *quotient*: either with $X = Y = Z = E$, or $X = Y = E$ and $Z = \mathbb{P}(E)$. Two functions *quotient* and *quotientP* are defined and the formal definition in the B set theory is:

$$
\begin{array}{l}
quotient \in (E \leftrightarrow E) \ \times \ (E \leftrightarrow E) \longrightarrow (E \leftrightarrow E) \\
\forall (R,S,T).(R \in E \leftrightarrow E \ \wedge S \in E \leftrightarrow E \ \wedge T \in E \leftrightarrow E \ \wedge R \subseteq quotient(S,T) \ \Rightarrow \ (T;R) \subseteq S) \\
\forall (R,S,T).\ (R \in E \leftrightarrow E \ \wedge S \in E \leftrightarrow E \ \wedge T \in E \leftrightarrow E (T;R) \subseteq S \ \Rightarrow \ R \subseteq quotient(S,T))
\end{array}
$$

The next lemma is useful, when using the relation *quotient* and it has been proved by the proof tool.

Lemma 3.1 $x \mapsto y \in quotient(S,T) \Leftrightarrow \forall z \cdot (z \mapsto x \in T \ \Rightarrow \ z \mapsto y \in S)$

The equivalence is split into two implications. In the first implication ($\Rightarrow$), the second property of *quotient*'s definition is enough. For the second implication ($\Leftarrow$), we have instantiated R (the second property of *quotient*'s definition) with the following set: $\{x \mapsto y \mid x \in E \wedge y \in E \wedge \forall z \cdot (z \mapsto x \in T \ \Rightarrow \ z \mapsto y \in S)\}$

DOMAIN is the set of pairs (e,e) where e is in the domain of the binary relation used as parameter and NOTDOMAIN is the set of pairs (e,e) where e is not in the domain of the binary relation used as parameter. The B definitions are:

$$
\begin{array}{l}
\text{DOMAIN} \in (E \leftrightarrow E) \longrightarrow (E \leftrightarrow E) \\
\forall R.(R \in E \leftrightarrow E \\
\quad \Rightarrow \ \text{DOMAIN}(R) = id(\mathsf{dom}(R))) \\
\text{NOTDOMAIN} : (E \leftrightarrow E) \leftrightarrow (E \leftrightarrow E) \\
\forall R.(R \in E \leftrightarrow E \\
\quad \Rightarrow \ \text{NOTDOMAIN}(R) = id(E) - id(\mathsf{dom}(R)))
\end{array}
$$

opt assigns to each binary relation R over E a binary relation modelling the criterion of optimality. S. Curtis defines *opt* as follow: $opt(R) = \in \cap quotientP(R, \ni)$. The $\in$ operator is not defined in B and first, we define it as a relation In.

$$
\begin{array}{l}
In \in \mathcal{P}(E) \leftrightarrow E \\
\forall (x,s).(x \in E \wedge s \in \mathbb{P}(E) \\
\quad \Rightarrow (s \mapsto x \in In \Leftrightarrow x \in s)) \\
opt \in (E \leftrightarrow E) \longrightarrow (\mathcal{P}(E) \leftrightarrow E) \\
\forall R.(R \in E \leftrightarrow E \\
\quad \Rightarrow \ opt(R) = In \ \cap \ quotientP(R, In^{-1}))
\end{array}
$$

The next operator is called *lambda* and it returns the image of each element in the domain of the relation; it is simply defined as the image of a singleton $\{x\}$ by the given relation:

$$
\begin{array}{l}
lambda \in (E \leftrightarrow E) \longrightarrow (E \longrightarrow \mathcal{P}(E)) \\
\forall (R,x).\ (R \in E \leftrightarrow E \ \wedge \ x \in \mathsf{dom}(R) \\
\quad \Rightarrow \ lambda(R)(x) = R[\{x\}])
\end{array}
$$

The greedy method is a method for computing optimal solutions in optimizations problems. The definition of the criterion for the local optimality should be stated. We assume that L defines the criterion for the local optimality and S defines the next possible step *opt* for the criterion of optimality. $greedy(L, S)$ is an operator defining pairs (x, y) built as descendants for S and optimal with respect to L.

$$
\begin{array}{l}
greedy \in (E \leftrightarrow E) \times (E \leftrightarrow E) \longrightarrow (E \leftrightarrow E) \\
\forall (L,S).(L \in E \leftrightarrow E \ \wedge \ S \in E \leftrightarrow E \\
\quad \Rightarrow \ greedy(L,S) = (lambda(S); opt(L)))
\end{array}
$$

The operator *greedy* satisfies a property derived with the proof tool.

Lemma 3.2 $\forall (L,S).(L \in E \leftrightarrow E \ \wedge \ S \in E \leftrightarrow E \ \Rightarrow \ greedy(L,S) = S \cap quotient(L, S^{-1}))$

The proof is discharged with the help of the prover; S. Curtis [9] writes that it is an useful property but she does not give any proof sketch. According to the definitions of $greedy(L, S)$ and $opt(L)$, we prove that $(lambda(S); In \ \cap \ quotientP(L, In^{-1})) = S \cap quotient(L, S^{-1})$ which is rewritten into $((lambda(S); In) \ \cap \ (lambda(S); quotientP(L, In^{-1}))) = S \cap quotient(L, S^{-1})$. The proof is split into two cases:

1. $((lambda(S); In) = S$ which is easy to prove and

2. $(lambda(S); quotientP(L, In^{-1})) = quotient(L, S^{-1})$ which is split in two inclusions

2.1. $(lambda(S); quotientP(L, In^{-1})) \subseteq quotient(L, S^{-1})$ which is easy to prove with *quotient* and *quotientP* definitions and

2.2. $quotient(L, S^{-1}) \subseteq lambda(S); quotientP(L, In^{-1}))$

For this last one we have used the first lemma: in one way ($\Rightarrow$) for $quotient(L, S^{-1})$ and the other one ($\Leftarrow$) for $quotientP(L, In^{-1})$.

This point shows clearly that the derivation of proofs of significant theorems is possible, if there is a mathematical expertise of the mathematical topics. The next operator is approaching an algorithmic idea of a process which is searching a value by repeating a step over a set as long as nothing is found. *rep* captures the idea of repeating a relation on a set as long as it is possible to apply the relation and the result of the application is simply a fixed-point. It behaves like a repeat-until loop and it may be operationally defined as follows: a pair (x, y) is in $rep(R)$, where R

is a binary relation over E, if either $x \notin \mathsf{dom}(R)$ and $x = y$, or $x \in \mathsf{dom}(R)$ and there is a path over R leading to $y \notin \mathsf{dom}(R)$. Formally, rep is defined as follows:

$$
\begin{array}{l}
rep \in (E \leftrightarrow E) \longrightarrow (E \leftrightarrow E) \\
\forall R.(R \in E \leftrightarrow E \\
\quad \Rightarrow rep(R) = \text{NOTDOMAIN}(R) \cup (R; rep(R))) \\
\forall (S,R).\ (\ R \in E \leftrightarrow E \wedge S \in E \leftrightarrow E \wedge \\
\quad \text{NOTDOMAIN}(R) \cup (R;S) \subseteq S \\
\quad \Rightarrow rep(R) \subseteq S)
\end{array}
$$

$repn$ (standing for n steps) computes the binary relation obtained by composing a given binary relation with respect to a given natural number.

$$
\begin{array}{l}
repn \in (E \leftrightarrow E) \times \mathbb{N} \nrightarrow (E \leftrightarrow E) \\
\forall S.(S \in E \leftrightarrow E \Rightarrow repn(S,0) = \mathsf{id}(E)) \\
\forall (S,n).(n \in \mathbb{N}^* \wedge S \in E \leftrightarrow E \\
\quad \Rightarrow repn(S,n) = (repn(S,n-1);S))
\end{array}
$$

The proof of theorem 3 (due to S. Curtis) requires intermediate lemmas on rep and $repn$. The first lemma is proved using the definition of rep and its minimality as a fixed-point (case $\subseteq$) and by induction (case $\supseteq$). The second lemma is a consequence of the first lemma and the two last lemmas are proved by induction.

Lemma 3.3 (i) $\forall S.(S \in E \leftrightarrow E \Rightarrow rep(S) = \text{UNION}(n) \cdot (n \in \mathbb{N} | (repn(S,n);\ \text{NOTDOMAIN}(S))))$

(ii) $\forall (S,x,y).(S \in E \leftrightarrow E \wedge x \in E \wedge y \in E \wedge x \mapsto y \in rep(S)$
$\Rightarrow \exists n.(n \in \mathbb{N} \wedge x \mapsto y \in repn(S,n) \wedge y \mapsto y \in \text{NOTDOMAIN}(S)))$

(iii) $\forall (S,n).(S \in E \leftrightarrow E \wedge n \in \mathbb{N} \Rightarrow (S; repn(S,n) = (repn(S,n);S))$

(iv) $\forall (S,n,m).(S \in E \leftrightarrow E \wedge n \in \mathbb{N} \wedge m \in \mathbb{N} \wedge m \leq n$
$quad \Rightarrow repn(S,n) = (repn(S,m); repn(S,n-m))$

The definitions provide a general framework for expressing optimization problems related to the greedy method; Curtis [9] defines four classes of greedy problems by characterizing conditions over theories. She specializes the general theory and we translate her characterizations in the B set theory. Using these characterizations, properties ensuring the optimality of the solution are proved.

3.2 Properties derived from the mathematical structures

Following Curtis [9], we prove four possible cases on mathematical structures which are ensuring the optimality of the solution for the greedy method. The four cases are related by implicative properties; the stronger case is called the better-local case and the weaker (or the most general) is the best-global)(Semantical diagram of Curtis's classification [9]). It is a simple rewriting of Curtis's theorem but they are mechanically checked by our proof engine and they confirm the results of Curtis.

The terminology for greedy algorithms mentions a construction step, a local optimality criterion and a global optimality criterion. Following the terminology, we consider four cases corresponding to assumptions made on the mathematical structures and the current problem. Following Curtis [9], every greedy algorithm that finds optimal solutions to optimisation problems complies with this principle. We do not discuss this aspect and use it like a postulate. Let assume that L and C are two preorders over a set E and S is a binary relation over E. *greedy* defines the greedy flavour of the method by combining both criteria over L for local and C for global. Curtis's theorems have the following schema:

$$\left(\begin{array}{l} preorder(L) \wedge\ preorder(C) \wedge \\ S \in E \leftrightarrow E\ \wedge H(L,C,S) \end{array}\right) \Rightarrow\ rep(greedy(L,S)) \subseteq\ (lambda(rep(S)); opt(C)))$$

On the theorem 1 (Best-Global), $H(L,C,S)$ equals to

$$\left(\begin{array}{l} \textsc{domain}(greedy(L,S)) = \textsc{domain}(S) \wedge \\ ((rep(S))^{-1}; greedy(L,S)) \subseteq (C; rep(S)^{-1}) \end{array}\right)$$

Our proof is quiet similar to Curtis's one.

On the theorem 2 (Better-Global), $H(L,C,S)$ equals to

$$\left(\begin{array}{l} \textsc{domain}(greedy(L,S)) = \textsc{domain}(S) \wedge \\ (S^{-1}; L; \textsc{domain}(S)) \subseteq (L; S^{-1}) \wedge \\ (S^{-1}; L; \textsc{notdomain}(S)) \subseteq L \wedge \\ (\textsc{notdomain}(S); L) \subseteq (C; rep(S)^{-1}) \end{array}\right)$$

Our proof is similar to Curtis's one; we prove that assumptions of the theorem 2 imply assumptions of theorem 1.

On the theorem 3 (Best-Local), $H(L,C,S)$ equals to

$$\left(\begin{array}{l} \textsc{domain}(greedy(C,S)) =\ \textsc{domain}(S) \wedge \\ \forall n.(n \in \mathbb{N}\ \Rightarrow \\ \qquad repn(greedy(L,S),n)\ \subseteq\ (lambda(repn(S,n)); opt(L))) \wedge \\ (S^{-1}; L; \textsc{notdomain}(S))\ \subseteq\ L \wedge \\ (\textsc{notdomain}(S); L)\ \subseteq\ (C; (rep(S))^{-1}) \end{array}\right)$$

Our proof is quiet similar to Curtis's one. We prove that assumptions of theorem 3 imply assumptions of theorem 1.

During the mechanical proof, we discover missing obvious cases and one obvious hypothesis.

On the theorem 4 (Best-Local),$H(L,C,S)$ equals to

$$\left(\begin{array}{l} \textsc{domain}(greedy(L,S)) = \textsc{domain}(S) \wedge \\ (S^{-1}; L; \textsc{domain}(S)) \subseteq (L; S^{-1}) \wedge \\ (S^{-1}; L; \textsc{notdomain}(S)) \subseteq L \wedge \\ (\textsc{notdomain}(S); L) \subseteq (C; rep(S)^{-1}) \end{array}\right)$$

Following Curtis's classification, we have proved the theorem in two manners: we prove that assumptions of theorem 4 imply assumptions of theorem 2 and that assumption of theorem 4 imply hypothesis of theorem 3. Our proofs are similar to Curtis's ones.

The number of interactions (clicks for choosing a function of the proof tool, choice of rules, ...) during the proof process for lemmas and theorems is presented 742 interactions with the prover.

3.3 Greedy theories

The four theories are related according to the diagram expressing the power of a theory with respect to another one and it tells us that the best global theory is the most general one. However, the three other ones provide a way to classify the greedy algorithms and they can be simpler to derive a proved step-by-step developed greedy algorithm. Each theory must ensure the existence of an optimal solution in the set of possible solutions and it is the key property of the proof of optimality of this algorithm. We will develop the greedy algorithm from the most general theory namely the best global theory. Our intuition is that the theorem 1 is a refinement proof and that the other theorems are instantiation proof and the corresponding development can be obtain from our first development.

4 Abstract algorithmic models for the greedy method

The mathematical framework is clearly defined and we have four classes for greedy problems; we develop a sequence of refined general models for the greedy method. The development process has the following steps:

- First abstract greedy model $BG0$
- Greedy refinement model $BG1$ for the Best-Global principle
- Refinement model for getting an algorithmic expression

We assume that E and rep, opt, $quotient$, $quotientP$, DOMAIN, NOTDOMAIN, $lambda$ and rep are given; they satisfy the properties of the previous defined theory ($repn$ can be defined later). Now, we should define several new constants:

(i) S is the step of the algorithm

(ii) C is the criterion for the global optimality and is a pre-order over E.

(iii) $greedy$ is the step of the greedy algorithm

(iv) $initial\_value$ is the initial value

(v) $preorder(O) \triangleq O \in E \leftrightarrow E \ \wedge \ \mathsf{id}(E) \subseteq O \ \wedge \ (O;O) \subseteq O$

MODEL
$BG0$
SETS
E
CONSTANTS
$C, S, lambda, rep, opt, initial\_value$
PROPERTIES
$\ldots$
$S \in E \leftrightarrow E$
$preorder(C)$
$initial\_value \in E$
VARIABLES
$solution$

INVARIANT
$solution \in E$
INITIALISATION
$solution :\in E$
EVENTS
$compute \ \widehat{=}$ ANY e WHERE
$e \in E$
$initial\_value \mapsto e \in (lambda(rep(S)); opt(C))$
THEN
$solution := e$
END
END

The first abstract model is not very surprising; it computes in one shot an optimal solution. We use a variable called $solution$. Two events are defined in the first abstract model. Initialisation is the initial event; it starts the execution of the abstract system by assigning any value to $solution$. $compute$ computes an optimal solution among the possible ones; the set of possible optimal solutions is not empty. The invariant of the system is simple: $solution \in E$; the assumptions on the constants allow us to derive the validity of the current model. There are no difficulties. This model states the initial problem and the next refinement will give some solution.

Now, we should choose one criterion and we choose to assume that the mathematical structure satisfies the Best-Global principle. We add the properties related to the principle and we introduce the function for modelling the repetition rep.

$$\text{DOMAIN}(greedy(L,S)) = \text{DOMAIN}(S) \ \wedge ((rep(S))^{-1}; greedy(L,S)) \ \subseteq \ (C; rep(S)^{-1})$$

The next property is derived from the assumptions over the current model.

Property 2 *Under the conditions defined by the greedy theory and the best global ones, the set* $(lambda(rep(S)); opt(C))$ *is not empty.*

The current model $BG0$ is refined by modifying the event $compute$, since the theorem 1 states that $rep(greedy(L,S)) \subseteq (lambda(rep(S)); opt(C))$. $compute$ computes an optimal solution among the possible ones; the set of possible optimal

solutions is not empty.

REFINEMENT
$BG1$
REFINES
$BG0$
CONSTANTS
L
PROPERTIES
$preorder(L)$
$\text{DOMAIN}(greedy(L,S)) = \text{DOMAIN}(S)$
$((rep(S))^{-1}; greedy(L,S)) \subseteq (C; rep(S)^{-1})$
VARIABLES
$solution$

INVARIANT
$solution \in E$
INITIALISATION
$solution :\in E$
EVENTS
$compute \mathrel{\widehat{=}} \text{ANY}\, e \text{ WHERE}$
$e \in E$
$initial\_value \mapsto e \in rep(greedy(L,S))$
THEN
$solution := e$
END
END

The new refinement $BG2$ introduces the effective computation step and the new invariant is much more elaborate. It includes the notion of existence of a solution while iterating. A new variable is introduced to contain the current value and it is called *current*. The new invariant states that the current value is an execution value leading to an optimal solution in finite time:

$$solution \in E \ \wedge current \in E \ \wedge$$
$$\exists n.(n \in \mathbb{N} \ \wedge \ initial\_value \mapsto current \in \ repn(greedy(L,S),n))$$

The initial event and the event *compute* are refined and a new event is introduced to model the step of the iteration. The final event is triggered, when the guard is true.

$compute \mathrel{\widehat{=}}$
WHEN
$current \mapsto current \in \text{NOTDOMAIN}(greedy(L,S))$
THEN
$solution := current$
END

The event *step* models the step of each computation while looping. We can replace BG1 properties (assumptions of theorem 1) by assumptions of theorem 2 (resp theorem 3 or theorem 4) to obtain a refinement proof, which is similar to the proof of theorem 2 (resp. theorem 3 or theorem 4). However, these developments seem to be independent, because the classification (of Curtis) is hidden inside the proof.

$step \mathrel{\widehat{=}}$
ANY e WHERE
$e \in E$
$current \mapsto e \in greedy(L,S)$
THEN
$current := e$
END
END

In this section, we explain how we can obtain other algorithms Best-Local, Better-Global and Better-Local by an instantiation of our previous model Best-Global. We give only an example. We omit to write the refinement model $BG3$ and we obtain a generic algorithm 2 from the refinement model $BG3$ by combining events. To obtain a complete development (similar to the previous one), we can instantiate name by name the previous development. Instantiation proof of refinement are obvious: both abstract models compute the same result. The other instantiation of proof obligation is that the properties of the Better-Global imply properties of the Best-Global system. This proof obligation is exactly the second theorem. We summarize the final statement of proofs discharged through the different refinement step.

Pre-condition:*C is a set of possible candidates*
Post-condition:*Either a solution S, or no solution does exist*

```
BEGIN
  solution::E;
  current:=initial_value;
  WHILE current in domain(S)
    DO
      current:= ChooseOneIn(opt(L)(S[{ current}]))
     OD
  solution:=current
END
```

Algorithm 2. General Greedy Algorithm

5 Conclusion

The incremental proof-based development of greedy algorithms is illustrated from the theoretical characterization of S. Curtis [9] and we state and check properties over mathematical structures related to the greedy method. The main advantage is to obtain a complete checking of Curtis's results, since a proof tool is not accepting results as *left to the reader* and every proof step should be completely discharged. The paper illustrates the use of generic developments based on very general mathematical structures; the genericity of proof-based development is a way to improve the proof process. This point should be developed by replaying the development of Prim's algorithm already developed using the refinement [4]; we plan to develop Kruskal's algorithm and other greedy problems. The fundamental question is also to be able to state what is the problem to solve and any development should start by the statement of mathematical structures and by the proof of properties required for the given problem (existence of solutions, for instance). Future work will develop new instantiations for the greedy-oriented developments.

References

[1] Abrial, J.-R., "The B-Book - Assigning Programs to Meanings", Cambridge University Press, 1996.

[2] Abrial, J.-R., *B#: Toward a synthesis between Z and B*, in: D. Bert and M. Walden, editors, *3nd International Conference of B and Z Users - ZB 2003, Turku, Finland*, Lectures Notes in Computer Science (2003).

[3] Abrial, J.-R. and D. Cansell, *Click'n'prove: Interactive proofs within set theory*, in: D. Basin and B. Wolff, editors, *16th Intl. Conf. Theorem Proving in Higher Order Logics (TPHOLs'2003)*, Lecture Notes in Computer Science **2758** (2003), pp. 1–24.

[4] Abrial, J.-R., D. Cansell and D. Méry, *Formal derivation of spanning trees algorithms*, in: D. Bert and M. Walden, editors, *3nd International Conference of B and Z Users - ZB 2003, Turku, Finland*, Lectures Notes in Computer Science (2003).

[5] Abrial, J.-R., D. Cansell and D. Méry, *Generic development of event based systems* (2002), internal report.

[6] Charlier, B., *The Greedy Algorithms Class: Formalization, Synthesis and Generalization*, Lectures Notes, UCL, Belgium (1995).

[7] ClearSy, *Web site B4free set of tools for development of B models*, `http://www.b4free.com/index.php` (2004).

[8] Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", MIT Press and McGraw-Hill, 2001.

[9] Curtis, S. A., *The classification of greedy algorithms*, Science of Computer Programming **49** (2003), pp. 125–157.

[10] Kaldewaij, A. and B. Schoenmakers, *Searching by elimination*, Science of Computer Programming **14** (1990), pp. 243–254.