On Proving Properties of Completion Strategies

Miki HERMANN
CRIN and INRIA-Lorraine
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy, France

e-mail: hermann@loria.crin.fr

Abstract

We develop methods for proving the *fairness* and *correctness* properties of rule based completion strategies by means of process logic. The concepts of these properties are formulated generally within process logic and then concretized to rewrite system theory based on transition rules. We develop in parallel the notions of *success* and *failure* of a completion strategy, necessary to support the proves of the cited properties. Finally we show the necessity of another property, called *justice*, in the analysis of completion strategies.

1 Introduction

The Knuth–Bendix completion procedure [KB70] presents a key tool for completing equational theories to confluent and terminating rewrite systems. Several properties were required to be fulfilled by the completion procedure with respect to its behavior and the produced result. The first attempt in this direction was Huet's *correctness* proof of the completion procedure [Hue81]. Huet also formulated the notion of *fairness* in completion in a certain way. It should be mentioned that Huet's presentation of the completion procedure differed considerably from the original presentation in [KB70]. Bachmair, Dershowitz, and Hsiang [BDH86] have put completion in an abstract framework, based on the notion of *transition rules*. The notions of *success* and *failure*, as well as the properties *fairness*, *soundness*, and *correctness* cristalised in the work of Bachmair [Bac87] and Dershowitz. Both give formulations of *correctness* and *fairness* in terms of equations and rewrite rules processed during the application of a transition rule based completion procedure. Moreover, Bachmair [Bac87] gives a characterization of *fairness* by several lemmas.

Fairness (a key property of completion procedures), as well as other eventuality properties, were treated in a more general framework of abstract processes and abstract programs [Fra86, GPSS80, Krö87, QS83].

An obvious tool for reasoning about programs and processes are several types of modal logic. Temporal logic [Krö87] is well-suited for reasoning about properties appearing during an execution of (mostly concurrent) fixed processes, but it has problems with from smaller units. Dynamic logic [FL79] copes perfectly with the problem of program composition from smaller units. Its drawback is that it can reason only about properties occurring before or after but not during the execution of a program. These problems were resolved in process logic [HKP82], which incorporates both temporal and dynamic logics.

We want to reason about properties of completion strategies (programs), composed from transition rules (basic instructions). Thus process logic is a suitable tool for this analysis. In this article we develop methods for proving the *fairness* and *correctness* properties of rule based completion strategies. The concepts of these properties are formulated generally within process logic and then concretized to rewrite system theory based on transition rules as given in [BDH86]. We develop in parallel the notions of *success* and *failure* of a completion strategy, necessary to support the proves of cited properties. Finally we show the necessity of another property, called *justice*, in the analysis of completion strategies. Unfortunately for us, the formalism we use is not capable to prove the justice of a strategy.

The proofs and several extensions and explications, not included into this article for lack of space, can be found in the research report [Her90].

2 Basic notation and definitions

The reader is supposed to be familiar with the concepts of term rewriting theory, temporal logic, dynamic logic, and process logic. For good overviews, see [DJ90] for rewrite systems, [Krö87] for temporal logic, [FL79] for dynamic logic, and [HKP82] for process logic.

Only to recall the few notations from rewrite systems theory: $\mathcal{T}(\mathcal{F},\mathcal{X})$ denotes the set of all terms (free algebra) over variables $\mathcal{X}$ and symbols $\mathcal{F}$, $Id_{\mathcal{T}(\mathcal{F},\mathcal{X})}$ denotes the set of all identities $t \simeq t$ over all terms $\mathcal{T}(\mathcal{F},\mathcal{X})$, $s \simeq t$ denotes an equation in E, $s \rightarrow t$ denotes a rewrite rule in a rewrite system R, $\longrightarrow_R$ denotes the smallest rewriting relation containing R, $s \succ t$ denotes the reduction ordering between the terms s and t, $t|_a$ denotes the subterm of t at a position $a \in \mathcal{P}os(t)$, $\mathcal{FP}os(t)$ denotes the set of all non-variable positions of the term t, $t\sigma$ denotes the substitution instance of the term t by the substitution σ, $CP(R_1, R_2)$ denotes the set of all critical pairs between the rules of rewrite systems R_1 and R_2, $\uparrow_R$ denotes the common ancestor relation $\overset{*}{\longleftarrow}_R \cdot \overset{*}{\longrightarrow}_R$ — *meetability*, $\downarrow_R$ denotes the common descendant relation $\overset{*}{\longrightarrow}_R \cdot \overset{*}{\longleftarrow}_R$ — *joinability.*

We remain basically in the scope of the process logic defined in [HKP82], and therefore we can use the axiomatization of this logic with the support of (strict) propositional dynamic logic [HR83]. The logic can be therefore called *strict simple process logic*, denoted StSiPL. For the exact syntax and semantics of this logic see [Her90].

The formula $\langle a \rangle\, p$ means in *some* executions of a, p is true; the formula $[a]\, p$ means in *all* executions of a, p is true. The formula $\mathbf{f}\, p$ means p is true in the *first* state of a path. The formula $p\, \mathbf{suf}\, q$ means there exists a suffix x which satisfies q and all suffixes y, where $x \prec y$, satisfy p (p is true *until* q becomes true). The formula $\mathbf{n}\, p$ means *next* state in the path exists and satisfies p. The formula $\mathbf{some}\, p$ means there exists a suffix satisfying p, i.e. p is true *somewhere*. The formula $\mathbf{all}\, p$ means all suffixes satisfy p, i.e. p is true *henceforth*. The formula $\mathbf{last}\, p$ says there the path is of finite length and the *last* state satisfies p. The formula **fin** (**inf**) says the path is of *finite* (*infinite*) length.

3.1 Process logic preliminaries

Define $\alpha \colon S \longrightarrow \mathcal{P}(\Sigma_0)$ to be the *applicability* function, an assignment of states to sets of atomic programs Σ_0 (programs that can be applied in the given state). For an atomic program $A \in \Sigma_0$ and a state $s \in S$ we have $A \in \alpha(s)$ if and only if there exists a state t, different from s, such that $(s, t) \in \tau(A)$.

Define now the predicate *apply* on atomic programs Σ_0 in a StSiPL model M. For a path $x \in S^\omega$ and an atomic program $A \in \Sigma_0$ we have $M, x \models \mathit{apply}(A)$ if and only if $A \in (\alpha \circ \mathit{first})(x)$, which determines if the atomic program A is applicable to the path x.

Define $\gamma \colon S \times S \longrightarrow \mathcal{P}(\Sigma_0 \cup \{\mathit{skip}\})$ to be the *state connectivity* function. $\gamma(s, t)$ determines the set of all atomic programs (or the *skip*) that transform the state s into the state t. For an atomic program $A \in \Sigma_0$ and states $s, t \in S$ we have $A \in \gamma(s, t)$ if and only if $(s, t) \in \tau(A)$. Moreover, for all states $s \in S$ the *skip* program is contained in $\gamma(s, s)$ according to the fact that *skip* does not do anything.

The state connectivity γ is *simple* if for all states $s, t \in S$ the set $\gamma(s, t)$ contains at most one element: $|\gamma(s, t)| \leq 1$. We consider only simple connectivity in the sequel, otherwise we would have problems with locating non-ambiguously the use of atomic programs.

Define now the predicate *use* on atomic programs Σ_0 in a StSiPL model M. For a path $x \in S^\omega$ and an atomic program $A \in \Sigma_0$ we have $M, x \models \mathit{use}(A)$ if and only if $A \in \gamma(\mathit{first}(x), \mathit{second}(x))$, where $\mathit{second} = \mathit{first} \circ \mathit{next}$. The predicate *use* determines whether A was used in the path x as the first applied atomic program to arrive at $\mathit{next}(x)$. It is sometimes necessary to know the fact that an atomic program was used the last time in a path x. This is expressed by a derived predicate $\mathit{lastuse}(A) = \mathit{use}(A) \land \mathbf{n\,all}\, \neg \mathit{use}(A)$. It is clear that for all atomic programs A we have $M, x \models \mathit{lastuse}(A) \supset \mathit{use}(A)$ and $M, x \models \mathit{use}(A) \supset \mathit{apply}(A)$.

We need also the (polymorphic) predicate *empty* operating on sets, preferably on the set of equations E.

3.2 Transition rules

A *transition system* [QS83] is a triple $\mathcal{S} = (S, \Sigma_0, \vdash_{\Sigma_0})$, where S is a countable set of transition *states*, Σ_0 is a *finite* set of *transition rules*, and $\vdash_{\Sigma_0}$ is a set of binary relations on S in bijection with the transition rules Σ_0.

In the sequel we observe the transition system $\mathcal{KB} = (S_{KB}, KB, \vdash_{KB})$, where states S_{KB} are formed by pairs $(E; R)$ of equations E and rewrite rules R. The Knuth–Bendix completion procedure is based on the following set KB of six transition rules:

Delete: $(E \cup \{s \simeq s\}; R) \vdash (E; R)$
Compose: $(E; R \cup \{s \rightarrow t\}) \vdash (E; R \cup \{s \rightarrow u\})$ if $t \longrightarrow_R u$
Simplify: $(E \cup \{s \simeq t\}; R) \vdash (E \cup \{u \simeq t\}; R)$ if $s \longrightarrow_R u$
Orient: $(E \cup \{s \simeq t\}; R) \vdash (E; R \cup \{s \rightarrow t\})$ if $s \succ t$
Collapse: $(E; R \cup \{s \rightarrow t\}) \vdash (E \cup \{u \simeq t\}; R)$ if $s \longrightarrow_R u$ by $l \rightarrow r \in R$ with $s \mathrel{\rhd\!\!\!\cdot}\, l$
Deduce: $(E; R) \vdash (E \cup \{s \simeq t\}; R)$ if $s \simeq t \in CP(R, R) - E$

where $\rhd\!\!\!\cdot$ denotes a proper encompassment ordering. We write $(E; R) \vdash_{KB} (E'; R')$ if the latter may be obtained from the former by one application of a rule in KB.

much problems to use it for deducing only one critical pair at a time. The whole set of critical pairs $CP(R, R)$ is always generated at once. Thus the transition rule *Deduce* can be replaced in KB by "*Deduction:* $(E; R) \vdash (E \cup CP(R, R); R)$" with the operational equivalence *Deduction* = **while** *apply*(*Deduce*) **do** *Deduce* **od**.

The StSiPL model, corresponding to the transition system $\mathcal{KB}$, will be denoted by M_{KB}. A strict regular program based on the set KB of transition rules as atomic programs is called a *completion strategy* (or simply *strategy*). A path x corresponding to an execution of a completion strategy a is called a *completion path*.

3.3 Observed strategy

Taking advantage of the defined predicate *apply* (and *empty*), we can easily write the *KBc* completion strategy as a strict regular program based on the transition rules KB. First, to structure well the completion strategy we use the subprograms *rR* and *rE* to describe the reduction of all rules and equations, respectively.

```
program rR is
begin
  while apply(Compose) do Compose od;
  while apply(Collapse) do Collapse od
end
```

```
program rE is
begin
  while apply(Simplify) do Simplify od;
  while apply(Delete) do Delete od
end
```

The observed completion strategy has the form

```
program KBc(E) is
begin
  while apply(Delete) do Delete od;
  while ¬empty(E) do
        if apply(Orient) then while apply(Orient) do Orient; rR; rE od
                         else  fail
        fi;
        Deduction; rE
  od
end
```

To make the proofs concerning the strategy more convenient, we describe parts of the *KBc* strategy by subprograms. These are the orient loop and the loop body:

```
program ol is
begin
  while apply(Orient) do
        Orient; rR; rE
  od
end
```

and

```
program lb is
begin
  if apply(Orient) then ol
                   else  fail
  fi;
  Deduction; rE
end
```

Then the main loop can be written as

```
begin
  while $\neg empty(E)$ do $lb$ od
end
```

3.4 Term rewriting theory within process logic

Classical (finite) rewrite systems can be investigated under process logic and transition system formalism, too. In this case the rewrite rules R become the atomic programs Σ_0. A pair (of terms) (s,t) is contained in the interpretation of an atomic program (rewrite rule) $l \rightarrow r \in R = \Sigma_0$ if and only if there exists a nonvariable position $a \in \mathcal{FP}os(s)$ and a substitution σ, such that $s|_a = l\sigma$ and $t = s[r\sigma]_a$. The predicate *apply* is then equivalent to the presence of a redex in the first term (state) of a path.

The basic applied strategy (if we can speak of a certain strategy at all) is a nondeterministic choice of rewrite rules from R, denoted just by the symbol R. Thereafter the computation of a normal form (normalization) could be expressed as

$$norm(R) \;\; = \;\; \mathbf{while}\ \exists p((p \in R) \wedge apply(p))\ \mathbf{do}\ R\ \mathbf{od}$$

The fact that R is terminating is expressed just as $[norm(R)]\,\mathbf{fin}$, which follows naturally from the intuitive meaning that there are no infinite rewritings. The *diamond lemma* saying "*A terminating rewrite system R is confluent iff it is locally confluent*" is then expressed by the following process logic expression

$$[norm(R)]\,\mathbf{fin} \supset (LConf(R) \;\equiv\; Conf(R)) \tag{1}$$

using the predicates $Conf(R) = \uparrow_R \subseteq \downarrow_R$ and $LConf(R) = CP(R,R) \subseteq \downarrow_R$ for confluence and local confluence respectively. The set $CP(R,R)$ is, in principle, interpreted as the relation $\longleftarrow_R \cdot \longrightarrow_R$.

4 Properties of the completion strategy

4.1 Success of completion

Following the intuitive meaning, a path x is *unfailing* (*successful*) if during the computation, expressed by x, no *fail* instruction was used. The appropriate PL expression formalizing this fact is $M, x \models unfailing \equiv \mathbf{all}\,\neg use(fail)$.

Now, if the *fail* instruction was used somewhere during the completion path x of the strategy KBc, we have

$$M_{KB}, x \models \mathbf{some}\, use(fail) \equiv \mathbf{fin} \wedge \mathbf{last}\,(\neg empty(E) \wedge \neg apply(Orient)) \tag{2}$$

from which we deduce $M_{KB}, x \models \mathbf{some}\, use(fail) \supset \mathbf{fin} \wedge \mathbf{last}\,\neg empty(E)$, which is equivalent to

$$M_{KB}, x \models (\mathbf{fin} \supset \mathbf{last}\; empty(E)) \supset \mathbf{all}\,\neg use(fail) \tag{3}$$

From the structure of the KBc strategy we deduce

$$[KBc]\,(\mathbf{all}\,\neg use(fail) \supset (\mathbf{fin} \supset \mathbf{last}\; empty(E))) \tag{4}$$

the case of *KBc* we can write

$$unfailing \quad = \quad \mathbf{fin} \supset \mathbf{last}\, empty(E) \tag{5}$$

From (5), we derive immediately the following theorem:

Theorem 4.1 *A finite completion path x of the completion strategy KBc is* **successful** *(or* unfailing*) if and only if in the last state of x the set of equations E is empty.*

4.2 Correctness

The intuitive meaning of *correctness* is a predicate coupled to the notion of success. A strategy a is correct with respect to the predicate $P(a)$, if the validity of this predicate is implied by each successful and finite computation. In the PL formalism:

$$[a]\,(\mathbf{fin} \wedge unfailing \supset \mathbf{last}\, P(a)) \tag{6}$$

Within a completion strategy a the predicate $P(a)$ is expressed by $Conf(R)$, meaning that a completing strategy is correct with respect to the confluence of the produced rewrite rules R. That justifies the following theorem.

Theorem 4.2 *A completion strategy b is* **correct** *if and only if b produces a confluent rewrite system R whenever b finishes successfully.*

Applying the equality (5) to (6), we get the expression

$$M_{KB} \models [KBc]\,(\mathbf{fin} \wedge \mathbf{last}\, empty(E) \ \supset\ \mathbf{last}\, Conf(R))$$

for the correctness of the completion strategy *KBc*.

For proving the correctness of *KBc* we need a supporting lemma, which is useful also for proofs of other properties.

Lemma 4.3 *For all transition rules $A \in (KB - \{Deduction\}) \cup \{Deduce\}$ the following proposition is valid: $M_{KB} \models [\mathbf{while}\ apply(A)\ \mathbf{do}\ A\ \mathbf{od}]\,(\mathbf{fin} \wedge \mathbf{last}\, \neg apply(A))$*

Lemma 4.3 implies immediately the following two propositions:

$$M_{KB} \models [rR]\,\mathbf{fin} \tag{7}$$

$$M_{KB} \models [rE]\,\mathbf{fin} \tag{8}$$

With a little more effort it is possible to prove

Proposition 4.4 $M_{KB} \models [ol]\,\mathbf{fin}$.

We cannot have $M_{KB} \models [ol]\,\mathbf{last}\,\neg apply(Deduction)$ because this implies $M_{KB} \models [ol]\,\mathbf{last}\,(CP(R,R) = E)$, which implies $M_{KB} \models [ol]\,\mathbf{f}\,\neg apply(Orient)$ and this is definitely not possible.

$$M_{KB} \models [ol]\,\mathbf{last}\,((\neg \bigvee_{A \in KB - \{Deduction\}} apply(A)) \wedge apply(Deduction)) \tag{9}$$

what indicates that only the *Deduction* rule can be applied after *ol*. This implies immediately $M_{KB} \models \langle ml\rangle\,\mathbf{inf} \supset \langle ml\rangle\,\mathbf{all}\,\neg lastuse(Deduction)$ or else

$$M_{KB} \models [ml]\,\mathbf{some}\,lastuse(Deduction) \supset [ml]\,\mathbf{fin}$$

The implication $\models [a]\,\mathbf{fin} \supset [a]\,\mathbf{some}\,lastuse(A)$ is trivially satisfied for each atomic program (transition rule) A and program a in each model M, therefore we have

$$M_{KB} \models [ml]\,\mathbf{some}\,lastuse(Deduction) \equiv [ml]\,\mathbf{fin} \tag{10}$$

Assume that x_{KB} is a *finite* completion path of the strategy *KBc* and y its suffix. Assume further that z_1 is a suffix of y and $z_2 = next(z_1)$.

$$x_{KB} \quad = \quad s_1 \ldots \overbrace{s_y \ldots \underbrace{s_{z_1} \overbrace{s_{z_2} \ldots s_n}^{z_2}}_{z_1}}^{y}$$

The fact, that each use of the *Deduction* transition rule (at y) is followed by a sequence of *Simplify* rules and then by a sequence of *Delete* rules in the strategy *KBc*, can be expressed as

$$M_{KB}, y \models use(Deduction) \supset \mathbf{n}\,(use(Simplify)\,\mathbf{suf}\,(apply(Delete) \supset use(Delete))) \tag{11}$$

From the structure of the transition rule *Delete* we deduce immediately the implication $M_{KB}, z_2 \models use(Delete) \supset \exists e(e \in E \supset e \in Id_{\mathcal{T}(\mathcal{F},\mathcal{X})})$. This one implies further

$$M_{KB}, z_2 \models \mathbf{last}\,empty(E) \wedge \mathbf{all}\,use(Delete) \supset (E \subseteq Id_{\mathcal{T}(\mathcal{F},\mathcal{X})}) \tag{12}$$

From the structure of the completion strategy *KBc*, as well as from (11), follows

$$M_{KB}, z_1 \models lastuse(Simplify) \supset \mathbf{n}\,\mathbf{all}\,use(Delete) \tag{13}$$

Comparing (12) with (13) and using $\vdash (a \supset b) \supset (a \wedge c \supset b \wedge c)$ gives the implication

$$M_{KB}, z_1 \models \mathbf{last}\,empty(E) \wedge lastuse(Simplify) \supset \mathbf{n}\,(E \subseteq Id_{\mathcal{T}(\mathcal{F},\mathcal{X})}) \tag{14}$$

Applying $\vdash (p \supset q) \wedge (p \supset r) \equiv p \supset (q \wedge r)$ and $\vdash_{PL} \mathbf{n}\,(p \wedge q) \equiv \mathbf{n}\,p \wedge \mathbf{n}\,q$ on (11) and on the implication $M_{KB}, y \models use(Deduction) \supset \mathbf{n}\,(CP(R,R) \subseteq E)$, we get

$$\begin{aligned} M_{KB}, y \quad \models \quad & use(Deduction) \supset \mathbf{n}\,((CP(R,R) \subseteq E) \wedge \\ & (use(Simplify)\,\mathbf{suf}\,(apply(Delete) \supset use(Delete)))) \end{aligned} \tag{15}$$

Comparing (15) and (14) results in the implication

$$M_{KB}, y \models \mathbf{last}\,empty(E) \wedge lastuse(Deduction) \supset \mathbf{n}\,\mathbf{all}\,LConf(R) \tag{16}$$

pass from y to x_{KB} and we get

$$M_{KB}, x_{KB} \models \mathbf{all}\,\mathbf{last}\; \mathit{empty}(E) \wedge \mathbf{some}\; \mathit{lastuse}(\mathit{Deduction}) \supset \mathbf{some}\,\mathbf{n}\,\mathbf{all}\; \mathit{LConf}(R) \quad (17)$$

It is clear that $\mathbf{all}\,\mathbf{last}\,p$ is equivalent to $\mathbf{last}\,p$. We have further $\vdash_{PL} \mathbf{n}\,p \supset \mathbf{some}\,p$ and $\vdash_{PL} \mathbf{some}\,\mathbf{some}\,p \supset \mathbf{some}\,p$ which proves

$$M_{KB} \models \mathbf{some}\,\mathbf{n}\,\mathbf{all}\; \mathit{LConf}(R) \supset \mathbf{some}\,\mathbf{all}\; \mathit{LConf}(R)$$

Therefore (17) implies

$$M_{KB}, x_{KB} \models \mathbf{last}\; \mathit{empty}(E) \wedge \mathbf{some}\; \mathit{lastuse}(\mathit{Deduction}) \supset \mathbf{some}\,\mathbf{all}\; \mathit{LConf}(R) \quad (18)$$

From (10) we imply

$$M_{KB}, x_{KB} \models \mathbf{some}\; \mathit{lastuse}(\mathit{Deduction}) \equiv \mathbf{fin} \quad (19)$$

Comparing (18) and (19) results in

$$M_{KB}, x_{KB} \models \mathbf{fin} \wedge \mathbf{last}\; \mathit{empty}(E) \supset \mathbf{some}\,\mathbf{all}\; \mathit{LConf}(R)$$

Using $\vdash (a \wedge b \supset c) \equiv (a \wedge b \supset a \wedge c)$ and $\vdash_{PL} \mathbf{fin} \wedge \mathbf{some}\,\mathbf{all}\,p \supset \mathbf{last}\,p$ on the previous implication gives

$$M_{KB}, x_{KB} \models \mathbf{fin} \wedge \mathbf{last}\; \mathit{empty}(E) \supset \mathbf{last}\; \mathit{LConf}(R) \quad (20)$$

The use of a reduction ordering in the transition rule *Orient* subsumes the proposition $[\mathit{norm}(R)]\,\mathbf{fin}$ therefore (20) implies

$$M_{KB}, x_{KB} \models \mathbf{fin} \wedge \mathbf{last}\; \mathit{empty}(E) \supset \mathbf{last}\; \mathit{Conf}(R) \quad (21)$$

according to (1). The finiteness of x_{KB} is expressed already in (21), therefore (21) is valid for all completion paths x_{KB} of the strategy *KBc*. Therefore we can generalize (21) to

$$M_{KB} \models [\mathit{KBc}]\,(\mathbf{fin} \wedge \mathbf{last}\; \mathit{empty}(E) \supset \mathbf{last}\; \mathit{Conf}(R))$$

The last implication validates the following theorem.

Theorem 4.5 *The completion strategy KBc is correct.*

4.3 Fairness

Our notion of *fairness* follows, in principle, the ideas of [Fra86, GPSS80, Krö87]. The difference, or additional required property, is the *application determinacy* of strict regular programs. We require that a completion strategy $a \in \Sigma$ must be deterministic with respect to the application of the transition rules Σ_0. The definition of application determinacy is based on the notion of a deterministic program a in dynamic logic which assumes the termination of a.

If $\langle a \rangle (\mathbf{fin} \wedge \mathbf{last}\, apply(A)) \supset [a] (\mathbf{fin} \wedge \mathbf{last}\, apply(A))$ *then* a *is* deterministic *with respect to the application of* A*. The program* a *is* deterministic in application *(wrt* Σ_0*) if and only if it is deterministic wrt the application of all* $A \in \Sigma_0$*.*

If a *and* b *are both deterministic in application, then also* $a; b$ *is* deterministic in application.

If a *is deterministic in application, then also* **while** p **do** a **od** *is* deterministic in application.

If a *and* b *are both deterministic in application, then also* **if** p **then** a **else** b **fi** *is* deterministic in application.

We are ready now to define the *fairness* property in general:

Definition 4.7 *The program* $a \in \Sigma$ *is* **fair** *(wrt* Σ_0*) if it is* deterministic in application *and for all atomic programs* $A \in \Sigma_0$ *the expression* $[a] (\mathbf{inf} \supset (\mathbf{all\,some}\, apply(A) \supset \mathbf{all\,some}\, use(A)))$ *is valid.*

This definition expresses exactly the following intuitive property: if there is an atomic program A that can be *applied* infinitely many times during an infinite computation with deterministic application of atomic programs, then the atomic program A is actually *used* infinitely many times during that computation. The definition reflects the general fairness principle expressed by the statement

> *Everything which is enabled infinitely many times within an environment with deterministic application will eventually occur.*

We use the shorthand *fair* to express the fairness property and thus write $[a]\, fair$ to declare that the strategy a is fair.

The application determinacy of the program a with respect of its fairness is unavoidable. Consider a new completion strategy derived from *KBc*, where the deterministic transition rule *Deduction* is replaced by the nondeterministic one *Deduce*. This new strategy could diverge on the system $R = \{fgfx \rightarrow gfx, ggx \rightarrow x\}$ if the transition rule *Deduce* never choose the second rule for computing critical pairs. On the other hand, computing the critical pairs of the second rule can leed to a finite canonical system. This is of course in contradiction with the notion of fairness.

For the fairness proof of a completion strategy we need to know the mutual dependence of transition rules with respect to the states where they get enabled or disabled. This is expressed in the following fact by a positive and negative invariant matrices.

Fact 4.8 *The proposition* $M_{KB} \models [A] (\mathbf{f}\, apply(B) \supset \mathbf{last}\, apply(B))$ *is valid for the transition rules A and B according to the following table:*

	B					
A	*Delete*	*Compose*	*Simplify*	*Orient*	*Collapse*	*Deduction*
Delete		*valid*		*valid*	*valid*	*valid*
Compose	*valid*		*valid*	*valid*	*valid*	*valid*
Simplify		*valid*			*valid*	*valid*
Orient	*valid*	*valid*			*valid*	*valid*
Collapse	*valid*		*valid*	*valid*		*valid*
Deduction	*valid*	*valid*	*valid*	*valid*	*valid*	

rules A and B according to the following table:

	B					
A	*Delete*	*Compose*	*Simplify*	*Orient*	*Collapse*	*Deduction*
Delete	*valid*	*valid*	*valid*	*valid*	*valid*	
Compose	*valid*	*valid*	*valid*	*valid*	*valid*	*valid*
Simplify		*valid*	*valid*		*valid*	
Orient	*valid*			*valid*		
Collapse		*valid*			*valid*	*valid*
Deduction		*valid*			*valid*	*valid*

The positive and negative invariants on programs $a, b \in \Sigma$ can be extended in a straightforward way on the constructs $a; b$ and **while** p **do** a **od**.

We can prove now the application determinacy of the key parts of the completion strategy *KBc*.

Lemma 4.9 *The programs rR, rE, ol, and therefore also the completion strategy KBc are deterministic in application.*

We continue with the proof of the second part of the fairness condition.

Lemma 4.10 *If* $a \in \Sigma$ *is finite and deterministic in application, and* $[a]\,\mathbf{some}\,\mathit{use}(A)$ *is valid then* **while** q **do** a **od** *is fair wrt* $A \in \Sigma_0$.

Corollary 4.11 *The completion strategy KBc is fair wrt the transition rule Deduction.*

Assume that x_{lb} is a computation path of lb. From the structure of lb follows

$$M_{KB}, x_{lb} \models \mathbf{all}\,\neg\mathit{use}(\mathit{fail}) \supset \mathbf{some}\,\mathit{use}(\mathit{Orient})$$

which implies $M_{KB} \models [lb]\,(\mathbf{all}\,\neg\mathit{use}(\mathit{fail}) \supset \mathbf{some}\,\mathit{use}(\mathit{Orient}))$ from

$$M_{KB} \models [lb]\,(\mathbf{f}\,\mathit{apply}(\mathit{Orient}) \supset \mathbf{some}\,\mathit{use}(\mathit{Orient})) \tag{22}$$

The proposition $M_{KB} \models [lb]\,\mathbf{fin}$ follows from (7), (8), Proposition 4.4, and the finiteness of *Orient* and *Deduction*. Using $\vdash_{PL} [a]\,(\mathbf{fin} \wedge p) \supset [\mathbf{while}\ q\ \mathbf{do}\ a\ \mathbf{od}]\,(\mathbf{inf} \supset \mathbf{all}\,p)$ on (22) implies $M_{KB} \models [ml]\,(\mathbf{inf} \supset (\mathbf{all}\,\neg\mathit{use}(\mathit{fail}) \supset \mathbf{all}\,\mathbf{some}\,\mathit{use}(\mathit{Orient}))$. In general we have $M_{KB}, x \models \mathbf{inf} \supset \mathbf{all}\,\neg(\mathit{fail})$ from (2), therefore with the use of $\vdash (a \supset b) \wedge (a \supset (b \supset c)) \supset (a \supset c)$ we get $M_{KB} \models [ml]\,(\mathbf{inf} \supset \mathbf{all}\,\mathbf{some}\,\mathit{use}(\mathit{Orient}))$, or else

$$M_{KB} \models [KBc]\,(\mathbf{inf} \supset \mathbf{all}\,\mathbf{some}\,\mathit{use}(\mathit{Orient}))$$

Applying to it $\vdash (a \supset b) \supset (a \supset (c \supset b))$ we get

$$M_{KB} \models [KBc]\,(\mathbf{inf} \supset (\mathbf{all}\,\mathbf{some}\,\mathit{apply}(\mathit{Orient}) \supset \mathbf{all}\,\mathbf{some}\,\mathit{use}(\mathit{Orient})))$$

Therefore we proved

Lemma 4.12 *The completion strategy KBc is fair wrt the transition rule Orient.*

$$M_{KB} \models [KBc]\,\mathbf{all}\,(e \in CP(R,R) \supset \mathbf{some}\, e \in E) \tag{23}$$

Lemma 4.12 implies that all persistent equations are oriented into rules during an infinite completion by the strategy *KBc*:

$$M_{KB} \models [KBc]\,(\mathbf{inf} \supset \mathbf{all}\,(e \in E \supset \mathbf{some}\, e \notin E)) \tag{24}$$

A statement combined of (23) and (24) was presented as a fairness definition in [Bac87]. Using the statements (23), (24), and the Critical Pair Lemma [BDH86, KB70], we derive by means of proof ordering that for two terms s and t, equal in the equational theory E, an infinite completion by the strategy *KBc* will generate a state $(E_i; R_i)$ such that $s \downarrow_{R_i} t$ [BDH86]: $M_{KB} \models [KBc]\,(\mathbf{inf} \supset (\mathbf{f}\,(s =_E t) \supset \mathbf{some}\,(s \downarrow_R t)))$.

Now we prove that *KBc* is fair wrt the rest of transition rules. We need the following fairness lemma.

Lemma 4.13 *If $a \in \Sigma$ is deterministic in application and $[a]\,(\mathbf{inf} \supset \mathbf{all}\,(apply(A) \supset \mathbf{some}\, use(A)))$ is valid then a is fair wrt $A \in \Sigma_0$.*

Lemma 4.13 in connection with Fact 4.8 is the main tool for proving fairness of a completion strategy wrt the transition rules *Compose*, *Collapse*, *Simplify*, and *Delete*.

Lemma 4.14 *The completion strategy KBc is fair wrt the transition rules Compose, Collapse, Simplify, and Delete.*

We could have proved the fairness of a completion strategy wrt *Compose* and *Collapse* of a completion strategy where the subprogram *rR* would have the form

```
program rR is
begin
  while apply(Collapse) do Collapse od;
  while apply(Compose) do Compose od
end
```

The required additional effort would be to prove that each application of *Compose*, disabled by the use of *Collapse*, is replaced by an application of *Simplify*.

To summarize the effort of this section, we state the final theorem

Theorem 4.15 *The completion strategy KBc is fair (wrt KB).*

4.4 Justice

Soundness (a property local to transition rules, and therefore not dealt with here), *success*, *correctness*, and *fairness* are not the only properties to be observed within a completion strategy. We need also the property of *justice*.

Example 4.16 Let us study the completion strategy

```
begin
  while $apply(Delete)$ do $Delete$ od;
  while $\neg empty(E)$ do
        while $apply(Orient)$ do $Orient$; $rR$; $rE$ od;
        if $empty(E)$ then $Deduction$ else $fail$ fi;
        $rE$
  od
end
```

The presented strategy is perfectly correct and fair, but it applies the *Deduction* rule only if the set of equations E is empty after ol (i.e., all equations were oriented into rules), otherwise it fails. This failure could be premature because a critical pair e could have been produced by *Deduction* and oriented into a rewrite rule r by *Orient*, and this rule r could simplify the previously unorientable equation in E. Therefore the dummy strategy ds is *not justified*. It is reasonable to fail only if all critical pairs from already produced rewrite rules were generated and completely simplified (*Deduction* followed by rE) and none of the remained equations can be oriented, as it was done in the justified strategy KBc.

Now we can define formally the discussed property:

Definition 4.17 *The strategy $b \in \Sigma$ is* **justified** *if and only if b is fair and for all sets of equations E if the strategy b fails on E then every fair strategy $c \in \Sigma$ fails on E, too. Formally:*

$$[b]\,fair \wedge \forall E([b(E)]\,\mathbf{some}\,use(fail) \supset \forall c([c]\,fair \supset [c(E)]\,\mathbf{some}\,use(fail))) \qquad (25)$$

The justice expression (25) is not an expression in the StSiPL logic any more. For proving it we must use a more subtle variant of process logic than StSiPL. Also the *justice* principle of [MP83] must be modified to cope well with our intentions.

5 Conclusion

Using the process logic, we were able to formulate the *correctness* and *fairness* properties for transition rule based systems in general and proving them for a specific completion strategy KBc. During the *fairness* proof we formulated two lemmas suitable for proofs of the *fairness* property of an arbitrary completion strategy based on the transition rules KB. Moreover, we showed that the particular formulation of *fairness* for the transition rules KB, given by Bachmair and Dershowitz, can be derived from our general one. Finally, we described the necessity of another property, called *justice*, for the analysis of completion strategies.

Acknowledgment

I would like to thank Nachum Dershowitz and Pierre Lescanne for the encouragement to follow this research, and the discussions on the fairness problem in completion.

[Bac87] L. Bachmair. *Proof methods for equational theories*. PhD thesis, University of Illinois, Urbana Champaign, Illinois, 1987.

[BDH86] L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proceedings 1st IEEE Symposium on Logic in Computer Science (LICS'86), Cambridge, (Massachusetts, USA)*, pages 346–357, June 1986.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, chapter 6, pages 243–309. Elsevier, Amsterdam, 1990.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.

[Fra86] N. Francez. *Fairness*. Springer-Verlag, 1986.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM Symposium on POPL, Las Vegas*, pages 163–173, January 1980.

[Her90] M. Hermann. On proving properties of completion strategies. Research report 90-R-149, Centre de Recherche en Informatique de Nancy, 1990. To appear in *Proceedings of 4th Conference on Rewrite Techniques and Applications, Como (Italy)*.

[HKP82] D. Harel, D. Kozen, and R. Parikh. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Science*, 25:144–170, 1982.

[HR83] J.Y. Halpern and J.H. Reif. The propositional dynamic logic of deterministic, well-structured programs. *Theoretical Computer Science*, 27:127–165, 1983.

[Hue81] G. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and System Science*, 23(1):11–21, August 1981. Also as: Rapport 25, INRIA, 1980.

[KB70] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[Krö87] F. Kröger. *Temporal logic of programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[MP83] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *Proceedings 10th ACM POPL Symposium, Austin, (Texas, USA)*, pages 141–154, 1983.

[QS83] J.P. Queille and J. Sifakis. Fairness and related properties in transition systems - A temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983.