Effectively eliminating auxiliaries

Stijn de Gouw[1,2] and Jurriaan Rot[3,⋆]

[1] CWI, Amsterdam, The Netherlands
[2] SDL, Amsterdam, The Netherlands
[3] LIP, Université de Lyon, CNRS, Ecole Normale Supérieure de Lyon, INRIA, Université Claude-Bernard Lyon 1, Lyon, France

Abstract. Auxiliary variables are used in the intermediate steps of a correctness proof to store additional information about the computation. We investigate for which classes of programs auxiliary variables can be avoided in the associated proof system, and give effective translations of proofs whenever this is the case.

1 Introduction

Auxiliary variables aid verification by storing additional information about the computation. Widely used instances include the recording of computation histories and the explicit access to control points via program counters. Auxiliary variables were first used by Clint [7] to prove properties about coroutines. Owicki [19] and Howard [13] used auxiliaries for reasoning about concurrent programs. Apt [3, 1] used auxiliaries to obtain intermediate assertions that denote decidable sets, which is useful for runtime checking. Recent applications of auxiliary variables are found in the Java Modeling Language [5], where they are called ghost variables. The power of auxiliaries is further illustrated by the fact that Frank de Boer himself advocated their use [8].

Auxiliaries are used temporarily, in the intermediate steps of a correctness proof, by instrumenting the program with assignments. A rule by Owicki and Gries [20] removes auxiliaries in a later proof step. As argued by Clarke [6], this use of auxiliary variables breaks compositionality, since the program fragments in the premise of the rule are not strict subprograms of that in the conclusion. Compositionality is crucial for a modular, syntax-directed proof construction.

Naturally the question arises: can auxiliary variables be avoided? This is the case for while programs and for recursive programs, since they have relative complete proof systems that do not contain Owicki and Gries' rule. Clarke showed that auxiliaries can be avoided in correctness proofs of programs with so-called simple coroutines, and raised the question whether history variables are necessary for concurrent programs with shared variables. Lamport [16] showed

⋆ The research of the second author was carried out at Leiden University and CWI. The second author is supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

that the full power of histories is not needed, but that using auxiliary variables as program counters suffices. To the best of our knowledge, it remained open whether auxiliary variables can be eliminated entirely.

In this paper we investigate for which classes of programs auxiliary variables can be avoided, and aim to give effective translations of proofs whenever this is possible. Hoffmann and Pavlova [12] gave such a translation for while programs, and we previously announced (without proof) a similar result for recursive programs [9], where it was applied to a practical example. Here we give the full translation, relying on so-called adaptation rules.

We introduce a translation from proofs of disjoint parallel programs using auxiliary variables, to proofs that do not. Our proof system uses adaptation rules instead of Owicki/Gries' rule for auxiliary variables, and technically relies on existentially quantifying auxiliary variables in specifications. Thereby, we show that, contrary to what is suggested in [4], auxiliary variables are not needed for disjoint parallel programs in the presence of suitable adaptation rules.

For programs with shared variable concurrency, we show that auxiliary variables *are* essential, in the sense that the associated rule cannot be replaced by *any* set of adaptation rules. This answers the open question by Clarke [6] "whether there is a proof system similar to the one originally described by Owicki which does not require the use of history variables" and confirms Kleymann's intuition that this is not case [15].

2 Preliminaries

We first fix some notation. Throughout this paper, we consider the usual interpretation of Hoare triples $\{p\}\ S\ \{q\}$ with respect to *partial* correctness. We use a first-order assertion language. Given an assertion p, we denote its free variables by $\mathit{free}(p)$, substitution of a term t for a variable x in p by $p[x := t]$ and in a term t' by $t'[x := t]$. The variables in a term t are denoted by $\mathit{var}(t)$. For a statement S (statements will be defined in subsequent sections) we denote by $\mathit{var}(S)$ the variables occurring in S, and by $\mathit{change}(S)$ the variables that occur on the left-hand side of an assignment in S. Given a list of variables $\bar{z}$, the statement $(S)_{\bar{z}}$ is obtained from S by removing all assignments to variables in $\bar{z}$ (using **skip** if S becomes empty), and $\bar{z}|_S$ is the sublist of variables in $\bar{z}$ that occur in $\mathit{change}(S)$, i.e., $\bar{z}|_S = \bar{z} \cap \mathit{change}(S)$. We abuse notation by using set-theoretic operations on lists, as in the previous line.

2.1 Auxiliary variables

Auxiliary variables store information about the computation. Formally, they are defined as follows.

Definition 1. *A closed list of auxiliary variables $\bar{z}$ for a given program is a sequence of program variables that appear only in assignments of the form $u := x$, where u is a variable in $\bar{z}$.*

The following rule, introduced by Owicki and Gries [20], allows to eliminate auxiliary variables in order to obtain the intended correctness triple:

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ (S)_{\bar{u}}\ \{q\}}\ (\mathsf{OG})$$

where $\bar{u}$ is a closed list of auxiliary variables for S, and $\bar{u} \cap \mathit{free}(q) = \emptyset$.

2.2 Adaptation rules

Support for modular reasoning in Hoare logic requires *adaptation rules* to adapt the specifications of Hoare triples to a specific context. The simplest example of an adaptation rule is the usual consequence rule. Adaptation rules are furthermore essential for proof systems about recursive procedures [2]. We briefly recall several adaptation rules taken from [4] (Figure 1) and [18] (Rule OLD).

$$\frac{\{p\}\ S\ \{q\}}{\{p \wedge r\}\ S\ \{q \wedge r\}}\ (\mathsf{INV})$$
where $\mathit{free}(r) \cap \mathit{change}(S) = \emptyset$

$$\frac{\{p\}\ S\ \{q\}}{\{\exists l.p\}\ S\ \{q\}}\ (\exists\text{-}\mathsf{IN})$$
where $l \notin \mathit{free}(q)$ and $l \notin \mathit{var}(S)$

$$\frac{\{p\}\ S\ \{q\}}{\{p[l := t]\}\ S\ \{l := t\}}\ (\mathsf{SUBST})$$
where $l \notin \mathit{var}(S)$ and $\mathit{var}(t) \cap \mathit{change}(S) = \emptyset$

$$\frac{p_0 \to p_1, \{p_1\}\ S\ \{q_1\}, q_1 \to q_0}{\{p_0\}\ S\ \{q_0\}}\ (\mathsf{CONS})$$

Fig. 1. Adaptation rules

Rule INV provides a basic way to reason about assertions whose truth remains invariant under execution of S. Rule SUBST instantiates a logical variable. Rule $\exists$-IN allows weakening preconditions under certain conditions. Figure 2 shows an example derivation using some of these rules.

$$\frac{\dfrac{\{p\}\ S\ \{q\} \quad q \to \exists z.q}{\{p\}\ S\ \{\exists z.q\}}\ (\mathsf{CONS})}{\{\exists z.p\}\ S\ \{\exists z.q\}}\ (\exists\text{-}\mathsf{IN})$$

Fig. 2. Adding existential quantifiers

Definition 2 (Adaptation completeness). *A proof system for a class of programs (ranged over by S) is* adaptation complete *if for all p, q, p', q': if*

$$\forall S.\ \models \{p\}\ S\ \{q\}\ \textit{implies}\ \models \{p'\}\ S\ \{q'\}$$

then any derivation of $\{p\}\ S\ \{q\}$ *can be extended to a derivation of* $\{p'\}\ S\ \{q'\}$ *(by only* adding *rule applications to the derivation).*

Remark 1. In the definition of adaptation completeness, one usually restricts to satisfiable correctness triples. Partial correctness avoids this extra condition, since all correctness triples are satisfiable.

To obtain an adaptation complete proof system, we use an approach due to Olderog [18]. Given assertions p, q, r and a sequence of variables $\bar{x}$, consider the following assertion:

$$\forall \bar{y}(\forall \bar{z}(p \to q[\bar{x} := \bar{y}]) \to r[\bar{x} := \bar{y}]) \tag{1}$$

where $\bar{y}$ is a sequence of fresh variables (not occurring in p, q or r) of the same length as $\bar{x}$, and $\bar{z} = \mathit{free}(p, q) \backslash \{\bar{x}\}$. The crucial property of the assertion (1) is:

Lemma 1 (Olderog Adaptation completeness). *The assertion* (1) *is the weakest assertion* w *such that* $\models \{w\}\ S\ \{r\}$ *for all finitely based state transformers*[4] S *with* $\mathit{var}(S) = \bar{x}$ *and* $\models \{p\}\ S\ \{q\}$.

Proof. See the text above Proposition 4.1 in [18]. □

We use assertion (1) to define an adaptation rule:

$$\frac{p' \to (\forall \bar{y}(\forall \bar{z}(p \to q[\bar{x} := \bar{y}]) \to q'[\bar{x} := \bar{y}])) \qquad \{p\}\ S\ \{q\}}{\{p'\}\ S\ \{q'\}}\ (\mathsf{OLD})$$

with $\bar{z}$ and $\bar{y}$ defined as in (1), and $\bar{x} = \mathit{var}(S)$ the list of program variables.

By Lemma 1 it is straightforward to show that adding Rule OLD yields a proof system that is adaptation complete. In particular, Rule OLD satisfies the following two properties (not unexpectedly, given its adaptation completeness).

Lemma 2 (Other adaptation rules redundant). *The adaptation rules given in Figure 1 are derivable with Rule* OLD.

Lemma 3 (Collapsing consecutive applications of Rule OLD). *Multiple consecutive applications of Rule* OLD *can be replaced by a single application.*

3 While programs

While programs form the basic building blocks for all the other classes of programs defined in the next sections. The syntax of while programs is as follows.

$$S ::= \mathbf{skip} \mid u := t \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi} \mid \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}$$

Figure 3 shows the standard proof system introduced by Hoare [10].

[4] Intuitively, a state transformer is finitely based if it reads and writes finitely many variables. See [18] for a precise definition.

$$\{p\}\ \mathbf{skip}\ \{p\}\ (\mathsf{SKIP}) \qquad \frac{\{p \wedge \neg B\}\ S_1\ \{q\}, \{p \wedge B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}\ (\mathsf{COND})$$

$$\{p[u := t]\}\ u := t\ \{p\}\ (\mathsf{ASGN}) \qquad \frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}\ (\mathsf{LOOP})$$

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1; S_2\ \{q\}}\ (\mathsf{SEQ}) \qquad \frac{p_0 \to p_1, \{p_1\}\ S\ \{q_1\}, q_1 \to q_0}{\{p_0\}\ S\ \{q_0\}}\ (\mathsf{CONS})$$

Fig. 3. Proof system PW

The next theorem shows that we can translate proofs of while-programs that use auxiliary variables to proofs without. Every rule application in the original proof is substituted, without having to consider the context of the enclosing proof, by at most three rule applications in the new proof. The translation is syntactic (no new loop invariants have to be invented) and fully automatic.

Theorem 1 (Auxiliary variables redundant for while programs). *Let $\overline{z}$ be a closed list of auxiliary variables for a statement S. There is an effective translation from any proof in* PW + *Rule* OG *of $\{p\}\ S\ \{q\}$ into a proof of $\{\exists \overline{z}.p\}\ (S)_{\overline{z}}\ \{\exists \overline{z}.q\}$ in* PW.

Proof. The translation is defined by induction on the derivation. We proceed by a case distinction on the last proof rule applied in the derivation of $\{p\}\ S\ \{q\}$.

- Rule SKIP. The desired $\{\exists \overline{z}.p\}$ **skip** $\{\exists \overline{z}.p\}$ follows by Rule SKIP.
- Rule ASGN. Then $\{p\}\ S\ \{q\}$ has the form $\{p[u := t]\}\ u := t\ \{p\}$. First note that for any assertion p and term t:

$$(\exists u.p[u := t]) \to (\exists u.p)\,. \tag{2}$$

  Next, we distinguish two cases:
  1. u is an auxiliary variable, which implies $u \in \overline{z}$. Since $(u := t)_u$ is **skip**, we must find a derivation of $\{\exists \overline{z}.p[u := t]\}$ **skip** $\{\exists \overline{z}.p\}$. By Rule SKIP we have $\{\exists \overline{z}.p[u := t]\}$ **skip** $\{\exists \overline{z}.p[u := t]\}$. Thus the desired result follows from (2) by an application of Rule CONS.
  2. u is a program variable, which entails $u \notin \overline{z}$ and $var(t) \cap \overline{z} = \emptyset$. Then $\{(\exists \overline{z}.p)[u := t]\}\ u := t\ \{\exists \overline{z}.p\}$, by Rule ASGN, and since u does not occur in $\overline{z}$, by Rule CONS: $\{\exists \overline{z}.p[u := t]\}\ u := t\ \{\exists \overline{z}.p\}$.
- Rule SEQ. Then $\{p\}\ S_1\ \{r\}$ and $\{r\}\ S_2\ \{q\}$ are derivable (for some intermediate assertion r). The induction hypothesis gives $\{\exists \overline{z}.p\}\ (S_1)_{\overline{z}}\ \{\exists \overline{z}.r\}$ and $\{\exists \overline{z}.r\}\ (S_2)_{\overline{z}}\ \{\exists \overline{z}.q\}$. If $(S_1)_{\overline{z}} =$ **skip** and $(S_1; S_2)_{\overline{z}} = (S_2)_{\overline{z}}$ (when S_1 consists of assignments to auxiliary variables) then the triple $\{\exists \overline{z}.p\}\ (S_1; S_2)_{\overline{z}}\ \{\exists \overline{z}.q\}$ follows by Rule CONS (similarly for the case $(S_2)_{\overline{z}} =$ **skip** and $(S_1; S_2)_{\overline{z}} = (S_1)_{\overline{z}}$). Otherwise $\{\exists \overline{z}.p\}\ (S_1; S_2)_{\overline{z}}\ \{\exists \overline{z}.q\}$ follows by Rule SEQ.
- Rule COND. Then $\{p \wedge B\}\ S_1\ \{q\}$ and $\{p \wedge \neg B\}\ S_2\ \{q\}$ are derivable, and $var(B) \cap \overline{z} = \emptyset$ (since auxiliaries do not occur in guards). By the induction

hypothesis: $\{\exists \bar{z}.(p \wedge B)\}\ (S_1)_{\bar{z}}\ \{\exists \bar{z}.q\}$ and $\{\exists \bar{z}.(p \wedge \neg B)\}\ (S_2)_{\bar{z}}\ \{\exists \bar{z}.q\}$. Because $var(B) \cap \bar{z} = \emptyset$, applying Rule CONS yields $\{(\exists \bar{z}.p) \wedge B\}\ (S_1)_{\bar{z}}\ \{\exists \bar{z}.q\}$ and $\{(\exists \bar{z}.p) \wedge \neg B\}\ (S_2)_{\bar{z}}\ \{\exists \bar{z}.q\}$. Finally, by applying Rule COND we obtain $\{\exists \bar{z}.p\}\ (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi})_{\bar{z}}\ \{\exists \bar{z}.q\}$.

- Rule LOOP. Then $\{p \wedge B\}\ S\ \{p\}$ is derivable and $var(B) \cap \bar{z} = \emptyset$. By the induction hypothesis: $\{\exists \bar{z}.(p \wedge B)\}\ (S)_{\bar{z}}\ \{\exists \bar{z}.p\}$. Because $var(B) \cap \bar{z} = \emptyset$, applying Rule CONS yields $\{(\exists \bar{z}.p) \wedge B\}\ (S)_{\bar{z}}\ \{\exists \bar{z}.p\}$. Rule LOOP yields $\{\exists \bar{z}.p\}\ (\textbf{while } B \textbf{ do } S \textbf{ od})_{\bar{z}}\ \{(\exists \bar{z}.p) \wedge \neg B\}$.
- Rule CONS. Then $p_0 \to p_1$ and $q_1 \to q_0$ are valid, and $\{p_1\}\ S\ \{q_1\}$ is derivable. By the induction hypothesis: $\{\exists \bar{z}.p_1\}\ (S)_{\bar{z}}\ \{\exists \bar{z}.q_1\}$. Observe that for any assertion p and q, if $p \to q$ is valid, then so is $(\exists \bar{z}.p) \to (\exists \bar{z}.p)$. Hence from $p_0 \to p_1$ and $q_1 \to q_0$ we may deduce $(\exists \bar{z}.p_0) \to (\exists \bar{z}.p_1)$ and $(\exists \bar{z}.q_1) \to (\exists \bar{z}.q_0)$. Applying Rule CONS we obtain the desired $\{\exists \bar{z}.p_0\}\ (S)_{\bar{z}}\ \{\exists \bar{z}.q_0\}$
- Rule OG. Then $\{p\}\ S\ \{q\}$ is derivable, $\bar{u}$ is the list of auxiliaries used in the application of the rule, and q does not contain any variables from $\bar{u}$. Our goal is to prove $\{\exists \bar{z}.p\}\ ((S)_{\bar{u}})_{\bar{z}}\ \{\exists \bar{z}.q\}$. Applying the induction hypothesis with $\bar{u}, \bar{z}$ as the auxiliaries yields $\{\exists \bar{u}\bar{z}.p\}\ (S)_{\bar{u}\bar{z}}\ \{\exists \bar{u}\bar{z}.q\}$. Clearly $(\exists \bar{z}.p) \to (\exists \bar{u}\bar{z}.p)$, and since q does not contain $\bar{u}$ we also have $(\exists \bar{u}\bar{z}.q) \to (\exists \bar{z}.q)$, thus our goal follows from Rule CONS. □

By instantiating Theorem 1 to the empty sequence of auxiliaries, we obtain:

Corollary 1 (Auxiliary variables redundant for while programs). *There is an effective translation from any proof in* PW *+ Rule* OG *of* $\{p\}\ S\ \{q\}$ *into a proof of* $\{p\}\ S\ \{q\}$ *in* PW.

4 Recursive programs

Programs with recursive procedures consist of a set of procedure declarations $D = \{P_i :: S_i \mid\ 1 \leq i \leq n\}$ and a main-statement S, where P_i is a procedure name and S_i and S are statements extending while-programs (Section 3) with:

$$S ::= P_i$$

which denotes a call to procedure P_i. The corresponding proof system requires a new ingredient: recursive procedures are proven correct *under a set of assumptions* about procedures. The assumptions are later discharged in a proof rule for procedure calls. Consequently, the statements that we derive with the proof system are not Hoare triples, but quadruples of the form $A \vdash \{p\}\ D|S\ \{q\}$ where A is a set of assumptions about the procedures $P_1, \ldots, P_n$ (i.e. Hoare triples $\{p_i\}\ P_i\ \{q_i\}$), and $D|S$ is a statement that can use the procedures declared in D (we omit D if it is clear from the context). Formally this requires adding sets of assumptions to all proof rules for while-programs given in Figure 3, but since these changes are obvious (the rules are independent of the assumptions and do not manipulate them), we omit them. Two rules are introduced for reasoning about calls (Figure 4). Rule ASMP shows how we can use the assumptions. Rule

CALL discharges the assumptions, provided that we can prove the procedure body S_i and the main-statement S using them. Besides these two new rules, to obtain a complete proof system, it turns out that the adaptation rules introduced in Section 2 (with additionally a set of assumptions) are also needed [2]. This leads us to the following formal definition of the proof system.

Definition 3 (Proof system for recursion). *Proof system* PR *consists of*

- *The rules in Figure 4,*
- *The rules from* PW *(Figure 3) under a set of assumptions,*
- *The adaptation rules (Figure 1) under a set of assumptions.*

$$\frac{}{A \vdash \{p\}\ D|S\ \{q\}\ (\mathsf{ASMP})} \text{ if } (\{p\}\ D|S\ \{q\}) \in A$$

$$\frac{A \vdash \{p_i\}\ D|S_i\ \{q_i\} \text{ for } i=1,\ldots,n \quad A \vdash \{p\}\ D|S\ \{q\}}{\vdash \{p\}\ D|S\ \{q\}}\ (\mathsf{CALL}) \quad \text{where } D = \{P_i :: S_i \mid 1 \le i \le n\}$$

Fig. 4. New proof rules in proof system PR

For recursive programs, the definition of *change*(...) is extended in the obvious way: *change*$(D|S)$ is the set of variables changed in S or any of the procedures called by S (declared in D). Along the same lines, $\bar{z}|_{(D|S)}$ is the sublist of variables from $\bar{z}$ changed by S or the procedures called by S. Furthermore, $(D|S)_{\bar{z}}$ is the program obtained from the statement S and procedure declarations D by removing all assignments to variables in $\bar{z}$ from S and the procedure bodies in D. The next definition uses these concepts to translate specifications.

Definition 4 (Translating specifications). *Given a set of n Hoare triples $A = \{\{p_i\}\ S_i\ \{q_i\} \mid i = 1, \ldots, n\}$ and a closed list of auxiliary variables $\bar{z}$, we define the translation TRANS$(A, \bar{z})$ by*
$A = \{\{\exists \bar{z}|_{(D|S_i)}.p_i\}\ (D|S_i)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S_i)}.q_i\} \mid i = 1, \ldots, n\}$.

The above translation requires no creativity to find appropriate procedure specifications; it can be performed fully mechanically. Using the new specifications, the below theorem shows that auxiliary variables can be avoided in proofs, deleting the auxiliaries from the main statement and all procedure bodies.

Theorem 2 (Removing auxiliaries). *Let $\bar{z}$ be a closed list of auxiliary variables for a recursive program $D|S$. There is an effective translation from any proof in* PR *+ Rule* OG *of $A \vdash \{p\}\ D|S\ \{q\}$ into a proof of TRANS$(A, \bar{z}) \vdash \{\exists \bar{z}|_{(D|S)}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q\}$ in* PR.

Proof. The translation is defined by induction on the derivation, with a case analysis on the last proof rule applied in the derivation of $A \vdash \{p\}\ D|S\ \{q\}$. For readability we omit D if it is clear from the context.

– Rule SKIP. We need to prove $\{\exists \bar{z}|_{\mathbf{skip}}.p\}$ **skip** $\{\exists \bar{z}|_{\mathbf{skip}}.p\}$ which is trivial, since $\bar{z}|_{\mathbf{skip}}$ is empty.
– Rule ASGN. Then $\{p\}\ S\ \{q\}$ has the form $\{p[u := t]\}\ u := t\ \{p\}$, and we need to give a proof of $\{\exists \bar{z}|_{u\,:=\,t}.[u := t]\}\ u := t\ \{\exists \bar{z}|_{u\,:=\,t}.p\}$. If $u \in \bar{z}$ then $\bar{z}|_{u\,:=\,t} = u$ and $\{\exists u.p[u := t]\}$ **skip** $\{\exists u.p\}$ is derived as in the proof of Theorem 1. If $u \notin \bar{z}$ then $\bar{z}|_{u\,:=\,t}$ is empty, so no translation is necessary.
– Rule SEQ. Then $\{p\}\ S_1\ \{r\}$ and $\{r\}\ S_2\ \{q\}$ are derivable. By the induction hypothesis, we get $\{\exists \bar{z}|_{S_1}.p\}\ (S_1)_{\bar{z}}\ \{\exists \bar{z}|_{S_1}.r\}$ and $\{\exists \bar{z}|_{S_2}.r\}\ (S_2)_{\bar{z}}\ \{\exists \bar{z}|_{S_2}.q\}$. Now since $\mathit{change}(S_i) \subseteq \mathit{change}(S_1; S_2)$ and $\bar{z} \cap (S_1; S_2)_{\bar{z}} = \emptyset$, by Figure 2 we get $\{\exists \bar{z}|_{S_1;S_2}.p\}\ (S_1)_{\bar{z}}\ \{\exists \bar{z}|_{S_1;S_2}.r\}$ and $\{\exists \bar{z}|_{S_1;S_2}.r\}\ (S_2)_{\bar{z}}\ \{\exists \bar{z}|_{S_1;S_2}.q\}$. If $(S_1)_{\bar{z}} =$ **skip** and $(S_1; S_2)_{\bar{z}} = (S_2)_{\bar{z}}$ (when S_1 consists of assignments to auxiliary variables) then $\{\exists \bar{z}|_{S_1;S_2}.p\}\ (S_1; S_2)_{\bar{z}}\ \{\exists \bar{z}|_{S_1;S_2}.q\}$ follows by Rule CONS (similarly for the case $(S_2)_{\bar{z}} =$ **skip** and $(S_1; S_2)_{\bar{z}} = (S_1)_{\bar{z}}$). Otherwise, $\{\exists \bar{z}|_{S_1;S_2}.p\}\ (S_1; S_2)_{\bar{z}}\ \{\exists \bar{z}|_{S_1;S_2}.q\}$ follows by Rule SEQ.
– Rule COND, LOOP and CONS are treated similarly to the proof of Theorem 1. For COND we use Figure 2 in the same way as in the above treatment of SEQ to extend the proofs of $\{\exists \bar{z}|_{S_i}.p_i\}\ (S_i)_{\bar{z}}\ \{\exists \bar{z}|_{S_i}.q_i\}$ for $i \in \{1, 2\}$ that come from the induction hypothesis to proofs of $\{\exists \bar{z}|_{S}.p_i\}\ (S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.p_i\}$ where $S =$ **if** B **then** S_1 **else** S_2 **fi**.
– Rule INV. Then $\{p\}\ S\ \{q\}$ is derivable, and r is an assertion with $\mathit{free}(r) \cap \mathit{change}(S) = \emptyset$. By the induction hypothesis we infer $\{\exists \bar{z}|_{S}.p\}\ (S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q\}$. Applying the invariance rule gives $\{(\exists \bar{z}|_{S}.p) \wedge r\}\ (S)_{\bar{z}}\ \{(\exists \bar{z}|_{S}.q) \wedge r\}$. Since $\bar{z}|_S$ and $\mathit{free}(r)$ are disjoint, we have for any assertion p: $(\exists \bar{z}|_{S}.p \wedge r) \leftrightarrow ((\exists \bar{z}|_{S}.p) \wedge r)$, thus Rule CONS yields $\{\exists \bar{z}|_{S}.p \wedge r\}\ (S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q \wedge r\}$.
– Rule ∃-IN. Then $\{p\}\ D|S\ \{q\}$ is derivable and l does not occur in S, D and q. The induction hypothesis gives us $\{\exists \bar{z}|_{S}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q\}$. Since l also does not occur in $(D|S)_{\bar{z}}$ we apply Rule ∃-IN: $\{\exists l.\exists \bar{z}|_{S}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q\}$. Finally, the consequence rule gives $\{\exists \bar{z}|_{S}.\exists l.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q\}$.
– Rule SUBST. Then $\{p\}\ D|S\ \{q\}$ is derivable, l does not occur in D or S and $\mathit{var}(t) \cap \mathit{change}(S) = \emptyset$. By the ind. hypothesis: $\{\exists \bar{z}|_{S}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{S}.q\}$. From Rule SUBST: $\{(\exists \bar{z}|_{(D|S)}.p)[l := t]\}\ (D|S)_{\bar{z}}\ \{(\exists \bar{z}|_{(D|S)}.q)[l := t]\}$. Since $\bar{z}|_{(D|S)}$ only contains variables that are changed, it is disjoint from l and $\mathit{var}(t)$, thus for any formula p we have the equivalence $(\exists \bar{z}|_{(D|S)}.p)[l := t] \leftrightarrow (\exists \bar{z}|_{(D|S)}.p[l := t])$. Hence, Rule CONS gives the desired correctness formula $\{\exists \bar{z}|_{(D|S)}.p[l := t]\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q[l := t]\}$.
– Rule ASMP. Then $(\{p\}\ D|S\ \{q\}) \in A$. The definition of $\mathrm{TRANS}(A, \bar{z})$ implies that $(\{\exists \bar{z}|_{(D|S)}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q\}) \in \mathrm{TRANS}(A, \bar{z})$. Therefore $\mathrm{TRANS}(A, \bar{z}) \vdash \{\exists \bar{z}|_{(D|S)}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q\}$ follows from Rule ASMP.
– Rule CALL. Then $A \vdash \{p_i\}\ D|S_i\ \{q_i\}$ for $i = 1, \ldots, n$ and $A \vdash \{p\}\ D|S\ \{q\}$ are derivable. The induction hypothesis gives us $\mathrm{TRANS}(A, \bar{z}) \vdash \{\exists \bar{z}|_{(D|S)}.p_i\}\ (D|S_i)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q_i\}$ for $i = 1, \ldots, n$, and $\mathrm{TRANS}(A, \bar{z}) \vdash \{\exists \bar{z}|_{(D|S)}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q\}$. Thus we can apply Rule CALL to obtain the desired $\vdash \{\exists \bar{z}|_{(D|S)}.p\}\ (D|S)_{\bar{z}}\ \{\exists \bar{z}|_{(D|S)}.q\}$.
– Rule OG. Then $\{p\}\ S\ \{q\}$ is derivable, $\bar{u}$ is the list of auxiliaries used in the application of the rule, and q does not contain $\bar{u}$. Our goal is to prove

$$\{\exists \bar{z}|_{(S)_{\bar{u}}}.p\}\ ((S)_{\bar{u}})_{\bar{z}}\ \{\exists \bar{z}|_{(S)_{\bar{u}}}.q\}.$$

An application of the induction hypothesis with $\bar{u}, \bar{z}$ as the auxiliaries gives us $\{\exists \bar{u}\bar{z}|_S.p\}\ (S)_{\bar{u}\bar{z}}\ \{\exists \bar{u}\bar{z}|_S.q\}$. Since $change((S)_{\bar{u}}) \subseteq change(S)$ we have $(\exists \bar{z}|_{(S)_{\bar{u}}}.p) \to (\exists \bar{u}\bar{z}|_S.p)$. Since q does not contain $\bar{u}$ we also have $(\exists \bar{u}\bar{z}|_S.q) \to (\exists \bar{z}|_{(S)_{\bar{u}}}.q)$, thus our goal follows from Rule CONS. □

Example 1. In [9], we proved the correctness of two sorting algorithms: Counting sort and Radix sort. Radix sort relies on an external sorting algorithm (Counting sort, for instance), and for its correctness it is crucial that the external sorting algorithm is *stable*, which means that equal elements in the input array appear in the same order in the output array. We formalized stability using an auxiliary array variable *idx* that keeps track of the original index in the input array of each element in the output array. This proves correctness with respect to an external (stable) sorting algorithm that updates *idx* appropriately. We would like to apply Theorem 2 (which appeared in a slightly different form in [9], without proof) to eliminate *idx* from the program, showing that Radix sort is correct whenever the external sorting algorithm is stable (without having to update *idx*). This is almost possible; the only small technical issue is that we assumed our assertion language to be first-order, while the translation of Theorem 2 relies on existentially quantifying the auxiliary (array) variable *idx*, thus we need second-order quantification. We leave a careful treatment of eliminating auxiliary array variables for future work.

From Theorem 2, we obtain an analogue of Corollary 1 for recursion.

Corollary 2 (Auxiliary variables redundant for recursive programs). *There is an effective translation from any proof in* PR *+ Rule* OG *of* $A \vdash \{p\}\ D|S\ \{q\}$ *into a proof of* $A \vdash \{p\}\ D|S\ \{q\}$ *in* PR.

5 Disjoint parallel programs

The syntax of disjoint parallel programs extends the syntax of while programs with a parallel operator:

$$[S_1||\ldots||S_n]$$

for any $n \geq 2$, syntactically restricted to statements $S_1, \ldots, S_n$ that are *disjoint*, which means that $change(S_i) \cap var(S_j) = \emptyset$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

The semantics of the parallel operator is modeled as usual by interleaving. The main proof rule for dealing with the parallel operator is as follows [11, 4]:

$$\frac{\{p_i\}\ S_i\ \{q_i\} \text{ for } i = 1 \ldots n}{\{\bigwedge_{i=1}^n p_i\}\ [S_1||\ldots||S_n]\ \{\bigwedge_{i=1}^n q_i\}}\ (\mathsf{PDJ})$$

where for all i, j with $i \neq j$: $free(p_i, q_i) \cap change(S_j)) = \emptyset$.

Adding the above Rule PDJ to PW does not yield a satisfactory proof system, as shown by the next result (Exercise 7.9 in [4]).

Theorem 3 (Incompleteness of PW + Rule PDJ). *The triple*

$$\{x = y\}\ [x := x + 1 || y := y + 1]\ \{x = y\}$$

is not provable in PW + *Rule* PDJ.

Proof. Suppose for a contradiction that $\{x = y\}\ [x := x + 1 || y := y + 1]\ \{x = y\}$ has a proof. This proof must include an application of Rule PDJ:

$$\frac{\{p_1\}\ x := x + 1\ \{q_1\} \qquad \{p_2\}\ y := y + 1\ \{q_2\}}{\{p_1 \wedge p_2\}\ [x := x + 1 || y := y + 1]\ \{q_1 \wedge q_2\}} \tag{3}$$

The only possible way that the proof can continue is by an application of Rule CONS, so the formulas below must be valid:

$$x = y \to p_1 \wedge p_2\,, \tag{4}$$

$$q_1 \wedge q_2 \to x = y\,. \tag{5}$$

By the premise of the rule application (3), we have

$$p_i \to q_i[x := x + 1] \text{ for } i \in \{1, 2\}\,. \tag{6}$$

In particular, we have $p_1[x := y] \to q_1[x := y + 1]$. But $p_1[x := y]$ is valid by (4), and thus

$$q_1[x := y + 1]$$

is valid. Instantiating y to $x - 1$ then yields the validity of q_1.

In a similar way, we derive the validity of q_2. But this means that $q_1 \wedge q_2$ is equivalent to **true**, which contradicts (5). □

The incompleteness result of Theorem 3 was introduced in [4] as a motivation for auxiliary variables.

Example 2. To see the use of auxiliary variables for disjoint parallel programs, we recall from [4] a proof of the triple $\{x = y\}\ [x := x + 1 || y := y + 1]\ \{x = y\}$ that uses an auxiliary variable together with Rule OG. Given a fresh variable z (i.e., $x \neq z$ and $y \neq z$), the correctness triples

$$\{x = z\}\ x := x + 1\ \{x = z + 1\} \qquad \text{and } \{y = z\}\ y := y + 1\ \{y = z + 1\}$$

are proved by Rule ASGN. Using Rule PDJ we get

$$\{x = z \wedge y = z\}\ [x := x + 1 || y := y + 1]\ \{x = z + 1 \wedge y = z + 1\}\,.$$

Now, consider the assignment $z := x$. Using Rule ASGN (and a simple application of Rule CONS) we get $\{x = y\}\ z := x\ \{x = z \wedge y = z\}$ and, using Rule SEQ:

$$\{x = y\}\ z := x; [x := x + 1 || y := y + 1]\ \{x = z + 1 \wedge y = z + 1\}$$

from which we derive

$$\{x = y\}\ z := x; [x := x + 1 || y := y + 1]\ \{x = y\}$$

by Rule CONS. Since z does not appear in the postcondition $x = y$, we may use Rule OG to obtain $\{x = y\}\ [x := x + 1 || y := y + 1]\ \{x = y\}$.

It turns out that auxiliary variables are not necessary in the presence of suitable adaptation rules. This is shown by the next result, which generalizes the translation given in Theorem 1 to disjoint parallel programs.

Theorem 4 (Auxiliary variables redundant for disjoint parallelism). *Let $\bar{z}$ be a closed list of auxiliary variables occurring in a disjoint parallel program S. There is an effective translation from any proof in* PW + *Rules* PDJ *and* OG *of $\{p\}\ S\ \{q\}$ into a proof of $\{\exists \bar{z}|_S.p\}\ (S)_{\bar{z}}\ \{\exists \bar{z}|_S.q\}$ in* PW + *Rules* PDJ, ∃-IN.

Proof. The proof is by induction on the derivation, similar to that of Theorem 2. The only remaining case (not treated in the proof of Theorem 2) is Rule PDJ.

– Rule PDJ. Then $\{p_i\}\ S_i\ \{q_i\}$ is derivable for $i \in \{1, \ldots, n\}$, and Rule PDJ is applied to get $\{\bigwedge_{i=1}^n p_i\}\ [S_1 || \ldots || S_n]\ \{\bigwedge_{i=1}^n q_i\}$. By the induction hypothesis we have proofs of $\{\exists \bar{z}|_{S_i}.p_i\}\ (S_i)_{\bar{z}}\ \{\exists \bar{z}|_{S_i}.q_i\}$ for $i \in \{1, \ldots, n\}$. Let $S = [S_1 || \ldots || S_n]$; by Figure 2 we obtain proofs of $\{\exists \bar{z}|_S.p_i\}\ (S_i)_{\bar{z}}\ \{\exists \bar{z}|_S.q_i\}$ for $i \in \{1, \ldots, n\}$. Now applying Rule PDJ yields

$$\{\textstyle\bigwedge_{i=1}^n (\exists \bar{z}|_S.p_i)\}\ (S)_{\bar{z}}\ \{\bigwedge_{i=1}^n (\exists \bar{z}|_S.q_i)\}\,. \tag{7}$$

For any $z \in \bar{z}|_S$, we have that z appears in exactly one of the S_i's, since the component programs are disjoint; say, in S_i. By the side-condition of the application of Rule PDJ in the original proof, we know that this means that z does not appear in any q_j with $j \neq i$. Therefore, we have the implication

$$(\bigwedge_{i=1}^n \exists \bar{z}|_S.q_i) \to \exists \bar{z}|_S.\bigwedge_{i=1}^n q_i\,.$$

Moreover, there is the easy implication $(\exists \bar{z}|_S.\bigwedge_{i=1}^n p_i) \to \bigwedge_{i=1}^n (\exists \bar{z}|_S.p_i)$. By (7), these two implications and Rule CONS, we conclude

$$\{\exists \bar{z}|_S.\textstyle\bigwedge_{i=1}^n p_i\}\ (S)_{\bar{z}}\ \{\exists \bar{z}|_S.\bigwedge_{i=1}^n q_i\}$$

as desired. □

Similar to the case of while and recursive programs, we obtain:

Corollary 3 (Auxiliary variables redundant for disjoint parallelism). *There is an effective translation from any proof in* PW + *Rules* PDJ, OG *of $\{p\}\ S\ \{q\}$ into a proof of $\{p\}\ S\ \{q\}$ in* PW + *Rules* PDJ, ∃-IN.

Example 3. We apply the translation of Theorem 4 to the proof of Example 2, choosing the empty sequence of auxiliaries (since there is no auxiliary to remove from the correctness triple we want to prove). The last rule application in that proof is Rule OG, which is translated to an application of Rule CONS:

$$\frac{\{\exists z(x = y)\}\ [x := x + 1 || y := y + 1]\ \{\exists z(x = y)\}}{\{x = y\}\ [x := x + 1 || y := y + 1]\ \{x = y\}}\ (\mathsf{CONS})$$

The proof of the correctness triple in the premise is a translation of the original proof of $\{x = y\}\ z := x; [x := x + 1||y := y + 1]\ \{x = y\}$ where the single variable z is chosen as the sequence of auxiliaries that are to be eliminated. It thus concludes with the translation of Rule CONS, with the premise (in the translated proof):

$$\{\exists z(x = y)\}\ [x := x + 1||y := y + 1]\ \{\exists z(x = z + 1 \wedge y = z + 1)\}\,.$$

This, in turn, arises from the translation of Rule SEQ, which concludes with an application of Rule CONS (since the assignment $z := x$ is eliminated), with premises:

$$\exists z(x = y) \to \exists z(x = z \wedge y = z) \text{ and} \quad (8)$$

$$\{\exists z(x = z \wedge y = z)\}\ [x := x + 1||y := y + 1]\ \{\exists z(x = z + 1 \wedge y = z + 1)\} \quad (9)$$

where (9) is derived from

$$\{x = z \wedge y = z\}\ [x := x + 1||y := y + 1]\ \{x = z + 1 \wedge y = z + 1\}$$

using Figure 2 (in the translation of Rule SEQ). The translated proof of the latter triple is the same as that of the original (see Example 2) since z does not appear in the statement.

6 Parallel programs with shared variables

Parallel programs extend while-programs with a parallel operator $[S_1|| \ldots ||S_n]$ for every $n > 1$ and an atomic region operator $\langle S\rangle$. Contrary to disjoint parallel programs considered in the previous section, here we make no assumptions on the statements appearing in $[S_1|| \ldots ||S_n]$; in particular, this allows shared variables between different S_i's. For instance, in the current setting, we allow the program

$$[x := x + 2||x := 0] \quad (10)$$

but the arguments of the parallel operator are not disjoint.

For the atomic region, we have the following rule:

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ \langle S\rangle\ \{q\}}\ (\mathsf{AT})$$

To reason about parallel composition, we use the notion of non-interference. Intuitively it expresses when an assertion is preserved by a given proof.

Definition 5 (Non-interfering proofs). *A proof of $\{p\}\ S\ \{q\}$ does not interfere with a proof of $\{p'\}\ S'\ \{q'\}$ if for all assertions r occurring outside of an atomic region in $\{p'\}\ S'\ \{q'\}$ and any sub statement T of S occurring outside an atomic region, we have $\{pre(T) \wedge r\}\ T\ \{r\}$, for any assertion $pre(T)$ occurring as a precondition of T in the proof of $\{p\}\ S\ \{q\}$.*

The proof rule for the parallel operator is:

$$\frac{\text{Non-interfering proofs of } \{p_i\}\ S_i\ \{q_i\} \text{ for } i = 1 \ldots n}{\{\bigwedge_{i=1}^{n} p_i\}\ [S_1 || \ldots || S_n]\ \{\bigwedge_{i=1}^{n} q_i\}}\ (\mathsf{PSV})$$

In [4] the premise of Rule PSV is formulated in terms of proof outlines. We refer to [4] for soundness and more details of these rules.

The proof system PW together with the above Rules PSV, AT is incomplete for the validity of correctness triples involving parallel programs with shared variables. Indeed, in [4, Lemma 8.6] it is shown that the triple

$$\{\mathbf{true}\}\ [x := x + 2 || x := 0]\ \{x = 0 \vee x = 2\}$$

is not provable using only PW + Rules PSV, AT. It is then shown that the above triple *is* provable using auxiliary variables together with Rule OG. In fact, the proof system PW + Rules PSV, AT, OG *is* complete [19].

One might expect that, similar to the treatment of disjoint parallelism in the previous section, we can again replace the rule for elimination of auxiliary variables by adaptation rules, while preserving completeness. However, as we show below in Theorem 5, that approach does not work: the proof system PW + Rules PSV,AT remains incomplete even with the addition of arbitrary adaptation rules. As explained in Section 2, the notion of "arbitrary adaptation rules" is captured precisely by adaptation completeness. Therefore, we use Rule OLD, which is adaptation complete for finitely based state transformers. (Whether it is adaptation complete for our parallel programs is open. Finitely based state transformers may be a larger class of programs than our parallel programs. Hence, for disjoint parallel programs there may be an adaptation rule which is stronger than Rule OLD.)

Theorem 5 (Auxiliaries needed for shared variable parallelism). *The triple*

$$\{\mathbf{true}\}\ [x := x + 2 || x := 0]\ \{x = 0 \vee x = 2\}$$

is not provable in PW *+ Rules* AT*,* PSV*,* OLD*.*

Proof. Assume that $\{\mathbf{true}\}\ [x := x + 2 || x := 0]\ \{x = 0 \vee x = 2\}$ has a proof in PW + Rules AT, PSV, OLD. We show that this leads to a contradiction. The proof must include an application of Rule PSV:

$$\frac{\{p_1\}\ x := x + 2\ \{q_1\} \qquad \{p_2\}\ x := 0\ \{q_2\}}{\{p_1 \wedge p_2\}\ [x := x + 2 || x := 0]\ \{q_1 \wedge q_2\}}\ (\mathsf{PSV}) \tag{11}$$

where the proofs of $\{p_1\}\ x := x + 2\ \{q_1\}$ and $\{p_2\}\ x := 0\ \{q_2\}$ are interference free. By Lemma 2 and Lemma 3 we can assume without loss of generality that the proof then concludes immediately, with a single application of Rule OLD:

$$\frac{\begin{array}{c}\mathbf{true} \rightarrow \forall y(\forall \bar{z}(p_1 \wedge p_2 \rightarrow (q_1 \wedge q_2)[x := y]) \rightarrow y = 0 \vee y = 2) \\ \{p_1 \wedge p_2\}\ [x := x + 2 || x := 0]\ \{q_1 \wedge q_2\}\end{array}}{\{\mathbf{true}\}\ [x := x + 2 || x := 0]\ \{x = 0 \vee x = 2\}}\ (\mathsf{OLD})$$

with $\bar{z} = \mathit{free}(p_1, p_2, q_1, q_2) \backslash \{x\}$. Instantiating the first premise with $x = 2, y = 4$ implies $\forall z((p_1 \wedge p_2)[x := 2] \to (q_1 \wedge q_2)[x := 4]) \to \mathbf{false}$, which is equivalent to

$$\exists \bar{z}(p_1[x := 2] \wedge p_2[x := 2] \wedge \neg(q_1 \wedge q_2)[x := 4])\,. \tag{12}$$

As we will see below, this leads to a contradiction with the side conditions and premises of the application (11), which we list first. Validity of the premises implies

1. $p_1 \to q_1[x := x + 2]$ and
2. $p_2 \to q_2[x := 0]$.

and the interference freedom conditions amount to the validity of

3. $p_1 \wedge p_2 \to p_1[x := 0]$,
4. $p_1 \wedge p_2 \to p_2[x := x + 2]$,
5. $q_1 \wedge p_2 \to q_1[x := 0]$ and
6. $q_2 \wedge p_1 \to q_2[x := x + 2]$.

(Note that $\bar{z}$ may occur in p_1, p_2, q_1 or q_2 and is implicitly universally quantified.) By (12) we may choose a valuation for $\bar{z}$ under which the following formulas hold:

7. $p_1[x := 2]$,
8. $p_2[x := 2]$ and
9. $\neg(q_1 \wedge q_2)[x := 4]$.

Together with the above validities, we derive (under the same valuation):

10. $q_1[x := 4]$ (by 1, 7),
11. $q_2[x := 0]$ (by 2, 8),
12. $p_1[x := 0]$ (by 3, 7, 8),
13. $q_2[x := 2]$ (by 6, 11, 12),
14. $q_2[x := 4]$ (by 6, 7, 13) and
15. $(q_1 \wedge q_2)[x := 4]$ (by 10, 14).

But 9 is in contradiction with 15 (note that we do not use 4 and 5). □

Remark 2. Theorem 5 strengthens [4, Lemma 8.6]: the latter is an incompleteness result for PW + Rules AT, PSV, CONS, but Rule CONS is subsumed by Rule OLD (see Section 2.2). For the proof system that includes OLD, the proof of [4, Lemma 8.6] immediately breaks, since it relies on the assumption that, in the proof assumed for a contradiction, the last applied rule is Rule CONS. In the presence of Rule OLD this assumption no longer holds, requiring a new proof.

Remark 3. Kleymann considers adaptation-complete proof systems for partial and total correctness of parallel programs in [14, 15]. In fact, the technical report [14] contains a proof of the program in Theorem 5, directly contradicting the theorem; however, the proof in [14] is invalid, neglecting crucial non-interference conditions in the application of Rule PSV. It does not appear in [15].

7 Conclusion and Future work

We have shown that for while programs, recursive programs and disjoint parallel programs, auxiliary variables are not needed and can be avoided using adaptation rules. We presented concrete translations of proofs, which means that no new method contracts and invariants need to be invented. The size of the produced proofs is linear in terms of the original proofs. For parallel programs with shared variables, auxiliary variables are essential. Table 1 summarizes the main results.

Class of programs	Proof system with auxiliaries	Proof system without auxiliaries
While	PW + Rule OG	PW
Recursion	PR + Rule OG	PR
Disjoint parallel	PW + Rules PDJ, OG	PW + Rules PDJ, ∃-IN
Parallel (shared var.)	PW + Rules AT, PSV, OG	*auxiliaries needed*

Table 1. Main results

It would be interesting to investigate the rôle of auxiliary variables for other classes of programs. One such class is programs that combine disjoint parallelism with recursion (cf. [17]). Of particular interest are object-oriented programs. A technical challenge there is that a naive translation of proofs that use fields as auxiliary variables introduces second-order quantification (over functions).

As Frank de Boer has experienced, *separation logic* [21] has emerged as the prime formalism for program correctness. We invite Frank to join us in our effort to extend our results to separation logic.

References

1. K. R. Apt. Recursive assertions and parallel programs. *Acta Inf.*, 15:219–232, 1981.
2. K. R. Apt. Ten years of Hoare's logic: A survey - part 1. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, 1981.
3. K. R. Apt, J. A. Bergstra, and L. G. L. T. Meertens. Recursive assertions are not enough - or are they? *Theor. Comput. Sci.*, 8:73–87, 1979.
4. K. R. Apt, F. S. de Boer, and E-R. Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 2009.
5. L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. Rustan M. Leino, and E. Poll. An overview of JML tools and applications. *STTT*, 7(3):212–232, 2005.
6. E. M. Clarke. Proving correctness of coroutines without history variables. *Acta Inf.*, 13:169–188, 1980.
7. M. Clint. Program proving: Coroutines. *Acta Inf.*, 2:50–63, 1973.

8. S. de Gouw, F. S. de Boer, W. Ahrendt, and R. Bubel. Weak arithmetic completeness of object-oriented first-order assertion networks. In *SOFSEM 2013: Theory and Practice of Computer Science, 39th International Conference, Proceedings*, pages 207–219, 2013.
9. S. de Gouw, F. S. de Boer, and J. Rot. Proof pearl: The KeY to correct and stable sorting. *J. Autom. Reasoning*, 53(2):129–139, 2014.
10. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
11. C. A. R. Hoare. Towards a theory of parallel programming. In *Operating System Techniques*, pages 61–71. Academic Press, 1972.
12. M. Hofmann and M. Pavlova. Elimination of ghost variables in program logics. In *Trustworthy Global Computing, Third Symposium, TGC 2007, Revised Selected Papers*, pages 1–20, 2007.
13. J. H. Howard. Proving monitors. *Commun. ACM*, 19(5):273–279, 1976.
14. T. Kleymann. Hoare logic and auxiliary variables. Technical Report ECS-LFCS-98-399, Laboratory for Foundations of Computer Science, University of Edinburgh, 1998.
15. T. Kleymann. Hoare logic and auxiliary variables. *Formal Asp. Comput.*, 11(5):541–566, 1999.
16. L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.
17. T. Nipkow. Hoare logics for recursive procedures and unbounded nondeterminism. In *Computer Science Logic, 16th International Workshop, CSL 2002, Proceedings*, pages 103–119, 2002.
18. E-R. Olderog. On the notion of expressiveness and the rule of adaption. *Theor. Comput. Sci.*, 24:337–347, 1983.
19. S. S. Owicki. *Axiomatic Proof Techniques for Parallel Programs*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1975.
20. S. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Inf.*, 6:319–340, 1976.
21. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), Proceedings*, pages 55–74, 2002.