On Refinement Calculus with Program Window Inference *

WANG Yunfeng LI Bixin PANG Jun ZHA Ming ZHENG Guoliang

(State Key Laboratory for Novel Software Technology ,Nanjiing University,Nanjiing 210093)
(Meteorology college , PLA University of Science and Technology , 211101)
wangyf@seg.nju.edu.cn

Abstract Data refinement is hard to deal with in a refinement tool than ordinary algorithmic refinement, since data refinement usually has to be done on a large program component at once. In the paper, firstly, the background is introduced, then the data refinement calculator is constructed. At last we argue an approach for data refinement which is based on data refinement calculus and program window inference.

Key words formal development method, data refinement calculus, program window inference

1 Introduction

Refinement is a process by which specifications are transformed into executable programs. The refinement calculus[1] is a formalization of the stepwise refinement approach, based on the weakest precondition calculus. The rules that encode useful program development principles are particularly important. One such principle is data refinement, where a data structure is replaced by another one and the total correctness of the program is preserved. Rules for data refinement make it possible to develop programs from specification s in just a small number of major refinement steps

Typical refinement transformations involve a considerable amount of formula manipulation which is error-prone. This is especially true for data refinement. Thus, tool support is needed for application of refinement calculus in practice. *Program Window Inference* [2]is a inference technique for refinement and proof tool. The idea is followed with *Window Inference*[3],for handling context in theorem proving.

Data refinement is hard to deal with in a refinement tool than ordinary algorithmic refinement, since data refinement usually has to be done on a large program component at once. In following sections, firstly, the background is introduced, then the data refinement calculator is constructed. At last we argue a approach for data refinement which is based on data refinement calculus and program window inference.

2 Basic Concept

2.1 Refinement Calculus

The refinement calculus is based upon the weakest preconditions of Dijkstra, which views programs as predicate transformers, i.e. functions from postconditions to precondition. The refinement calculus extends the guard command of Dijkstra with specification statement. It is a wide spectrum language and a set of correctness preserving rules for deriving executable program from specification. The emphasis on refinement comes from the observation that it is more effective to develop a program and its correctness proof together, as opposed to attempting to verify a given program retrospectively. It is a calculus because the transformation rules calculate the refined program. It includes a specification statement in addition to the usual executable constructs. This integration of specification and exection in one languange is the key to a smooth development process, since it allow a program to be developed by a series of transformation within a single language. The initial program is typically a specification statement, the final program contains only executable code and intermediate program are a mixture of the two.

- **Specification statement**

* The work here is supported by China National Science Foundation and “National 95 Project”(98-780-01-07-06).

One program statement that is particularly important in the refinement calculus is the *specification statement,* which provides a convenient way of embedding abstract specification into programs.. The *frame* of statement x:[pre, post] (x) is a variables list that can be updated, all other variables must remain unchanged. The specification statement id defined as:

$[\![x : [pre , post]]\!] P \equiv pre \wedge (\forall x \bullet post \Rightarrow P)$

2.2 Program Window Inference

The program window inference extends window inference to explicitly deal with program contexts[2]. The goal is to provide better support for mechanizing refinement. In program window inference the programs and predicates are separated and the explicit mechanisms for handling program context and program logic is argued. This allows one to reason more directly at the program level, without to reduce everything to predicates.

Even simple programs can build up significant refinement context, hence , handling the context in effective way is essential item for supporting program refinement. As most of the development of a program is done by refining its components, the program window inference theory provides good support for refinement of program component in context. Following introduces way to handle preconditions.

For a (traditional) window:

$$H \vdash \{P\}\ S \circ \{P\}\ S' \qquad (2.2\_1)$$

The equivalence program window is

$$H\ ;\ \textbf{pre}\ P \vdash S \circ S' \qquad (2.2\_2)$$

Where the annotation **pre** is part of syntax of program window, it represents the label of precondition.

Consider a selection command:

$$\{P\}\ \textbf{if}\ g\,1 \rightarrow S1 \quad g2 \rightarrow S2\ \textbf{fi} \qquad (2.2\_3)$$

In the refinement of S, it needs to make use of the precondition P and guard g1, thus these preconditions must become explicit, i.e. to replace (2.2_3) by

$$\{P\}\ \textbf{if}\ g\,1 \rightarrow \{P \wedge g1\}\ S1 \quad g2 \rightarrow \{P \wedge g2\} S2\ \textbf{fi} (2.2\_4)$$

This leads to replication of information, which is partly undesirable if P is a large formula. So a new type window, the program window is argued, which includes precondition along with ordinary hypotheses. In program window, the window opened for refinement of S1 in (2.2_3) is

$$H\ ;\ \textbf{pre}\ P \wedge g1 \vdash S1 \circ S1' \qquad (2.2\_5)$$

Where the precondition context of the selection command, **pre** P , is automatically augmented by the guard g1 to form the precondition context for refinement of S1.

3 Data Refinement Calculus

3.1 Data refinement

Data refinement can be viewed as a special case of program refinement, in which the abstract local variables of a program are replaced by the concrete set of variables, while the structure of the program remains largely unchanged.

Assuming that the state space of program S is variable to be refined (a) with some globe variables (g) and the state space of program S' is the concrete variables (c) with g, where a, c and g are all disjoint.

We choose any predicate transformer ***sim*** that takes predicate on the variables a, g to predicates on the variables c, g. For program P and P', P is data refined by P', written as $P \prec P'$.

Data refinement definition1 :

$$P \prec P' \text{ iff } \quad sim\ ;\ P \sqsubseteq P'\ ;\ sim \qquad (3.1\_1)$$

Where the operator ';' is functional compositional (of predicate transformers). In fact the above ***sim*** is a co-simulation.

If the sim is a simulation, noted as sim*, then the data refinement id defined as:

Data refinement definition2 :

$$P \prec P' \text{ iff } \quad P\ ;\ sim^* \sqsubseteq sim^*\ ;\ P' \qquad (3.1\_2)$$

Given the abstraction relation AI between abstract and concrete data structure, for any predicate φ over abstract variables, the semantics of sim and sim* can be defined in weakest precondition as follows:

$$[\![sim]\!]\ \varphi \triangleq \exists\ a \bullet AI \wedge \varphi \qquad (3.1\_3)$$

$$[\![sim^*]\!]\ \varphi \triangleq \forall a \bullet AI \Rightarrow \varphi \qquad (3.1\_4)$$

In practice, it is usuful to place some restrictions on **sim** to ensure that data refinement distribution through the various program constructors and thus leaves the

structure of the abstract program unchanged. For example, if the co-simulation sim is disjunctive and strict, then the data refinement relation $\prec$ satisfies the following rule:

$(S1 \prec S1') \wedge (S2 \prec S2') \Rightarrow S1;S2 \prec S1';S2'$ (3.1_5)

3.2 Calculation of data refinement

Since the scope of the local variable is a block statement, data refinement is modeled as a transformation of the whole block. In this way, data refinement becomes a special case of algorithm refinement: the refinement is just on the whole block.The block is noted as:

$|[\ \mathbf{var}\ a \mid Init \bullet P\ ]|$ (3.2_1)

which is the target of data refinement. Var a is local variable added into the program and is initialized according to the initialization predicate Init.

The semantics of the block is given in weakest precondition: for any predicate φ not containing free a,

$[\![|[\ \mathbf{var}\ a \mid Init \bullet P\ ]|]\!] \varphi \triangleq \forall a \bullet Init \Rightarrow [\![P]\!] \varphi$ (3.2_2)

- Data refinement calculator

For data refinement of the block, we assume that the abstract and the concrete variables are related by abstract relation AI, the data refinement calculator D_{AI} for predicates and $\mathcal{D}_{AI}$ for program statements is introduced, such that

$|[\ \mathbf{var}\ a \mid Init \bullet P]| \sqsubseteq |[\mathbf{var}\ c \mid D_{AI}(Init) \bullet \mathcal{D}_{AI}(P)\ ]|$ (3.2_3)

For a predicate φ, D_{AI} and its dual calculator D^*_{AI} are defined as:

$$D_{AI}(\varphi) \triangleq \exists\ a \bullet AI \wedge \varphi \qquad (3.2\_4)$$

$$D^*_{AI}(\varphi) \triangleq \forall a \bullet AI \Rightarrow \varphi \qquad (3.2\_5)$$

From 3.1 section and follow the concept of dual and adjoint of the predicate transformer in data refinement[4] we have:

$P \prec P'$ iff $sim\ ;\ P\ ;\ sim^* \sqsubseteq P'$ (3.2_6)

Thus, (sim ; P ; sim*)is defined as the least(most abstract)data refinement for statement P such that:

$$sim\ ;\ P\ ;\ sim^* \sqsubseteq \mathcal{D}_{AI}(P) \qquad (3.2\_7)$$

4 Program Window Inference with Data Refinement

In performing refinement of a component of a program, the context of the component is important. Window inference and program window inference introduced above provides an excellent approach to handle such contextual information. But the data refinement is not included into the approach. Data refinement is hard to deal with in a refinement tool than ordinary algorithmic refinement, since data refinement usually has to be done on a large program component at once.

Since the scope of the local variable is a block statement, data refinement is modeled as a transformation of the whole block. In this way, data refinement becomes a special case of algorithm refinement: the refinement is just on the whole block.

In this section, we argue a approach to handle data refinement with program window inference, which is based on data refinement calculator introduced in section 3.2.

4.1 Window Opening Rule

The data refinement is just on the block, thus only one window opening rule is required for the block. As the program can not operate on both the abstract and the concrete state at the same time, hence a data refinement transformation must replace all occurrences of the abstract variable in one step.

For data refinement on the block (3.2_3), we have the rule:

H ; **pre** P[c\c'] $\wedge D_{AI}$ (Init) ; **lval** L[c\c'] $\wedge c \in$ Var; **inv** Inv[c\c'] $\wedge c \in T' \wedge AI \vdash \boxed{P} \prec \mathcal{D}_{AI}(P)$

H ; **pre** P; **lval** L; **inv** Inv $\vdash$ $[\![|[\ \mathbf{var}\ a : T \mid Init \bullet \boxed{P}\ ]|]\!] \sqsubseteq [\![|[\mathbf{var}\ c : T' \mid D_{AI}(Init) \bullet \mathcal{D}_{AI}(P)\ ]|]\!]$ (4.1_1)

Where AI is the abstract invariant representing the relation between the abstract and concrete variables. With this rule, the block transformation is focus on calculation of the concrete commands, while the other complex predicates are collected into the **pre** and **inv**

context, such as initialization for concrete variable D_{AI} (Init) and AI.

The sembol $\prec$ represents data refinement relation between abstract and concrete command.(see section 3.1), and it is reflective and transitive. For getting the same relation in subwindow, ($\boxed{P} \prec D_{AI}$ (P)) can be written as D_{AI} ($\boxed{P}$) $\sqsubseteq$ P', and (4.1_1) is written as:

H ; **pre** P[c\c'] $\wedge D_{AI}$ (Init) ; **lval** L[c\c'] $\wedge c \in$ Var; **inv** Inv[c\c'] $\wedge c \in$ T' $\wedge$ AI $\vdash D_{AI}$ ($\boxed{P}$) $\sqsubseteq$ P'

H;**pre**P;**lval**L;**inv**Inv $\vdash$ |[**var** a:T|Init • $\boxed{P}$]| $\sqsubseteq$ |[**var** c:T' | D_{AI} (Init)• P']| (4.1_2)

This rule's form is same as the algorithm refinement rule.

4.2 Comparing with algorithm refinement rule

In this paper, data refinement is handled as the special algorithm. But it is different on window opening and focus transformation rule.

- **window opening rule**

For data refinement there is just one window opening rule that is on the variable declaration block. The focus is over all commands in the block, not their component, since the program can not operate on both the abstract and the concrete state at the same time, hence a data refinement transformation must replace all occurrences of the abstract variable in one step.

When focussing inside the block, the abstract invariant AI (noting the relation of abstract and concrete variable)and the predicates transformed such as D_{AI} (Init) are augmented into the context of the new window.

- **Focus transformation rule**

For algorithm refinement, the focus transformation is followed the refinement calculus rule of Morgan[5], which refines the statement into different construct (such as iteration, alteration, loop .)

For data refinement, the focus transformation does not change the structure of the command. It just calculates the concrete command from the abstract one by the data refinement calculator.

5 Conclusion

We have argued a approach for data refinement which is based on data refinement calculus and program window inference. The approach supports the calculation style of data refinement: a concrete program can be automatically constructed from the abstract one. The program window inference is effective way to manage complexity during refinement. Managing and providing access to program context can lead to simpler refinement with less replication. In the future work, we will research the special case of the data refinement calculator and simplified the data refinement mostly with program windows inference technique.

Reference

1. R. J. R. Back and von Wright. Refinement calculus, part I: Sequential programs. In REX Workshop for Refinement of Distributed Systems, volume 430 of Lecture Notes in Computer Science, Nijmegen, The Netherlands, 1989. Springer-Verlag.
2. Ray Nickson and Ian Hayes. Supporting Contexts in Program Refinement. Technical Report 96-29, Software Verification Research Centre, The University of Queensland, 1995.
3. Jim Grundy. A window inference tool for refinement. In Cliff B. Jones, Roger C. Shaw, and Tim Denvir, editors. Fifth Refinement Workshop, Workshops in Computing. BCS FACS, Springer-Verlag, 1992., pages 230--254.
4. J. von Wright. A Lattice-theoretica base for program refinement. PhD thesis, Abo Akademi University, SF-20500 Turku, Finland, Sept. 1990.
5. C. C. Morgan. Programming from Specifications. Prentice Hall International Series in Computer Science, 2nd edition, 1994.