Ordering strict partial orders to model behavioural refinement

Mathieu Montin
Université de Toulouse ; INP ; IRIT
2 rue Camichel, BP 7122
31071 Toulouse Cedex 7, France
mathieu.montin@enseeiht.fr

Marc Pantel
Université de Toulouse ; INP ; IRIT
2 rue Camichel, BP 7122
31071 Toulouse Cedex 7, France
marc.pantel@enseeiht.fr

ABSTRACT

Behavioural refinement plays a key role in the development of correct–by–construction complex systems such as real time distributed systems. Indeed, behavioural models coupled with formal methods allow to assess the correctness of system models with respect to system requirements. In that purpose, behavioural refinements allow preserving the correctness of the models during the development phases, from the early specification to the final embedded system. Refinement is usually handled in an operational matter such that each level of abstraction is derived from notions coming from the closest higher level of abstraction. This vision however is unsuitable when coupled with denotational semantics where the solutions are not built but rather validated by the semantics. Our work targets the definition of a refinement relation compatible with this kind of semantics. This relation is integrated to CCSL where refinement is not a native construct of the language and whose semantics is given in a denotational manner.

CCS Concepts

•**Software and its engineering** → **Formal software verification;** •**Computer systems organization** → *Embedded software;* •**Theory of computation** → *Type theory;*

Keywords

Refinement; Behavioural models; Agda; CCSL

1. INTRODUCTION

Software is now ubiquitous and involved in complex interactions between the human user and the physical world in so-called cyber-physical systems (CPS). To handle the growing complexity of these systems, separation of concerns is mandatory. Two different kinds of separation are usually identified throughout their development: the horizontal and vertical separation. The first one corresponds to the various concerns in the system architecture which might be described in different domain specific modelling languages (DSMLs). The second one corresponds to the different steps in a development leading from the requirements to the implementation through the use of refinements. The horizontal separation is usually handled through the abstraction of the different parts of the system in a common behavioural language. However, most of these languages allow the expression of constraints between the different preoccupations of the system but lack the required expressiveness to handle vertical separation. In this paper, we propose a formal definition of the relation of refinement in a denotational context. It relies on an order between the strict partial orders that are used to bind together the different instants on which events occur. This work has been conducted using the Agda proof assistant and associated to CCSL denotational semantics, although no knowledge regarding both languages will be needed here as details cannot be given due to space limits but the whole development is available on the first author's web page.

2. A REFINEMENT EXAMPLE

The transition system depicted in Figure 1 is an example of a simple system, which can be alternatively enabled and disabled. While it is active, an action can be executed any number of times. We focus on event traces and event refinement and not state traces and refinement like [3] as we model time using instantaneous event in a synchronous manner.

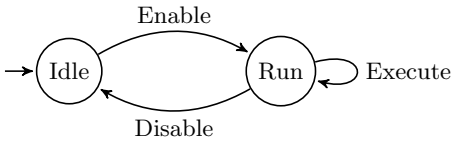

Figure 1: A simple system

By picturing our example with a transition system, we implicitly provided an operational semantics. Indeed, the transition system can be seen as a machine which builds a correct execution for the system. A denotational semantic however, does not give any way of creating such traces and instead provides predicates to assess the correctness of a given trace. This distinction is essential since most refinement strategies rely on operational semantics while we aim at handling systems through their denotational semantics. A possible trace for our example is depicted on Figure 2. t_{en}, t_{di} and t_{ex} respectively represent the occurrences of the "Enable", "Disable" and "Execute" transitions. This trace could have been generated from our transition system and would be validated by any denotational semantics describing our example.

This trace starts with the birth of the system and possibly goes on indefinitely, which makes this representation partial. In addition, this design places each event on the same timeline, thus ignoring horizontal separation. In order to make it visible, we represent each event on a specific timeline on

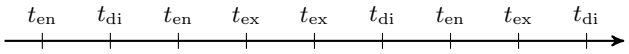

Figure 2: A trace on a single timeline

Figure 3. The instants on each timeline are totally ordered and those in the same vertical dotted lines are coincident. These notions will be elaborated when introducing the strict partial orders.

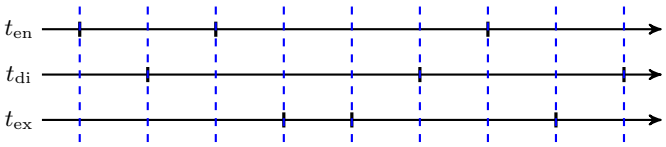

Figure 3: One timeline per event

The action executed by this system can be specified in various ways. In this paper, we imagine that it is connected to a light through the use of a memory containing a variable m. This variable will be assigned the values 1 or 0, and the light will be turned on and off accordingly. When the system is enabled (t_{en} transition), the light remains down until a button is pressed (t_{ex} transition) shuts it down. Pressing the same button will alternatively turn it off and on. Disabling the system (t_{di} transition) turns it off, as depicted on Figure 4.

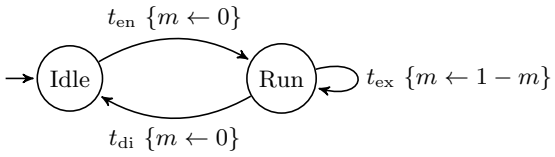

Figure 4: The system pilots a light

By specifying the system behaviour, we defined events to add to its traces. t_{m_0} and t_{m_1} respectively correspond to the variable m being assigned 0 and 1. These additions belong to horizontal separation since we added a new part to our system (the module linked to the light). One of these possible traces is depicted in Figure 5. As we add new events, refinement cannot be defined as a simple inclusion of traces like [5].

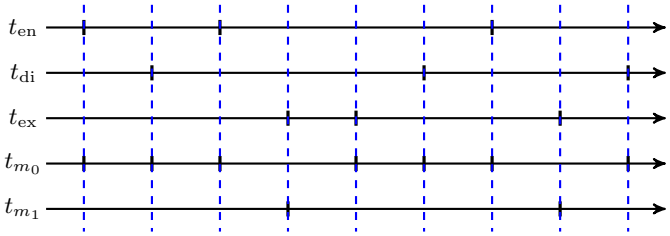

Figure 5: The new trace of the system

Some events are occurring simultaneously, for instance t_{en} always occurs on an instant coincident to an occurrence of t_{m_0}. Such relations between events can be defined in CCSL (a simple case of sub-clocking here), which has been handled in a previous work on the mechanization of this language.

It is important to note that when describing this system, we implicitly took a certain point of view regarding its definition. We deliberately ignored some low level concerns regarding the way such a memory is handled. This is a matter of vertical separation. The next sections of this paper will focus on a more concrete level of abstraction. But first we need to introduce some standard notions regarding time handling in asynchronous languages.

3. REPRESENTATION OF TIME

When considering executable languages, we observe different events which occur on given instants of time. Although the common vision of time is a straight line inducing a total order between each existing instant, asynchronous systems introduce uncertainties that weaken this order. Two instants are indeed not necessarily comparable, which leads to the use of partial orders to represent the existing links between them. Thus, each pair of instants is either:

- comparable, through a precedence relation $\prec$
- equivalent, through a coincidence relation $\approx$
- unrelated (neither comparable nor equivalent)

Some properties are required for these relations to form a strict partial order:

- $\approx$ is an equivalence relation
- $\prec$ is irreflexive regarding $\approx$
- $\prec$ is transitive
- $\prec$ respects the classes induced by $\approx$

4. BEHAVIOURAL REFINEMENT

4.1 Goal

Whenever a certain event occurs on a given instant, its occurrence is considered immediate and punctual in the timeline. However in some cases, such an event can be decomposed in smaller events which contradicts this property. In our example, the "Enable" event can be viewed as a sequence of sub–events, such as powering up the system, retrieving the address of m, computing the value of 0 (here there is no actual computation since 0 is an atomic value, but there could be in the case of a more complicated arithmetical expression) and storing this value at the right address. These events, except for the first one, are used to handle the computation and the storing of a value in a memory. Taking into account these events require to view the system at a more concrete level, in which case its representation as a transition system is depicted in Figure 6.

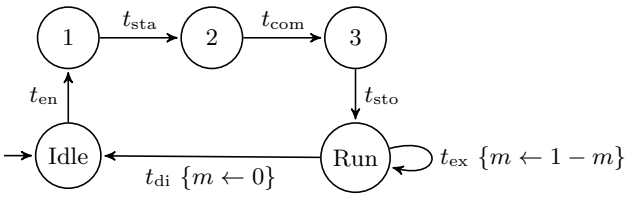

Figure 6: The refined system

The "Enable" transition has been refined in several transitions. t_{en} represents the powering of the system, t_{sta} the

stacking of the address of m, t_{com} the computing of the value of the expression 0 and t_{sto} the storing of the computed value at the stacked address.

Since both points of view we discussed are valid representations of our system, it should be possible to describe them in any concurrent language, without losing the link that binds them. This leads to the main goal of our proposal to model behavioural refinement, which is to describe a system at different levels of observation without losing the link between these levels. Thus, an instant at a certain level could be refined by several instants at a lower level, just like the "Enable" event was split into several different events.

Addressing this issue would allow system developers to focus on their specific view of the system rather than a common view shared among all of them. By representing it from the right angle, they could grasp their constraints even better without bothering about more concrete details. The system could then be solved at different levels with the guarantee that none of them will be compromising the others. Furthermore, this notion of refinement could be used to make explicit and prove simulations and bisimulations (or mostly weak bisimulations) between systems. In this case, the two specifications would not be different levels of observation of a system, but different ways of specifying its behaviour.

4.2 Different levels of refinement

In our example, the higher level of observation is represented on Figure 7 while the lower level is represented on Figure 8. In both these timelines, events not refined are omitted, for the sake of clarity. They do not influence the reasoning we are conducting, thus their omission is acceptable. The different instants have been annotated with natural numbers in order to manipulate them more easily. From the higher point of view, all the instants on which the subevents occur are equivalent to each other and to the containing event. Their underlying order has no impact on the trace of the system at this level.

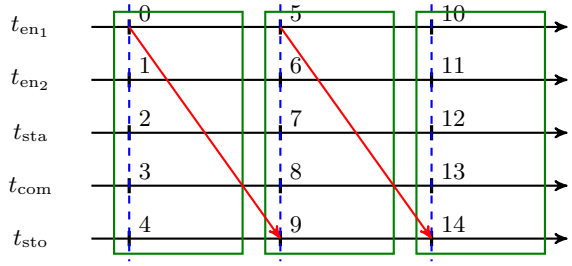

Figure 7: The annotated higher level of observation

For the lower level of observation, the different instants are ordered in such a way that they respect the specification in Figure 6. The vertical dashed lines represent the equivalence classes induced by the thinner strict partial order, while the rectangles represent the ones induced by the refined strict partial order.

The representation in Figure 7 allows us to to assess the coincidence and the precedence relation that bind its different instants, as subsets of $\mathbb{N} \times \mathbb{N}$. Since both these relations must be transitive, the coincidence must be symmetrical and they must form a strict partial order, we will omit the related

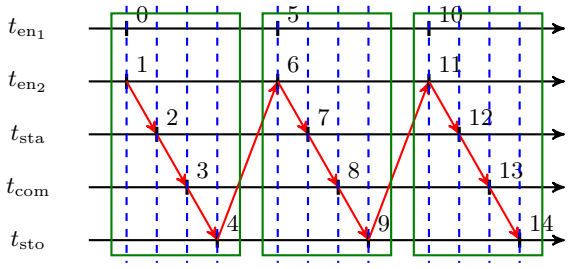

Figure 8: The annotated lower level of observation

elements which can be deduced from these properties.

Coincidence			Precedence
(0 , 1)	(0 , 2)	(0 , 3)	(0 , 5)
(0 , 4)	(5 , 6)	(5 , 7)	
(5 , 8)	(5 , 9)	(10 , 11)	(5 , 10)
(10 , 12)	(10 , 13)	(10 , 14)	

These traces are potentially infinite, thus we only give the visible subset of each relation. However, we can define them mathematically for any natural number in order to handle their infinite number. This is done by relying on the Euclidean decomposition by 5:

$$\forall (a, a') \in \mathbb{N}^2, \exists!\ (q, r, q', r') \in \mathbb{N}^4\ :\\ a = 5q + r \wedge r < 5 \wedge a' = 5q' + r' \wedge r' < 5$$

These relations are defined as follows:

$$\forall (a, a') \in \mathbb{N}^2, a \approx_2 a' \overset{d}{\iff} q = q'\\ \forall (a, a') \in \mathbb{N}^2, a <_2 a' \overset{d}{\iff} q < q'$$

The same work can be achieved for the lower level of observation, which is displayed on Figure 8. The relations extracted from Figure 8 are depicted in the table below. As previously explained, only the relevant couples are mentioned.

Coincidence	Precedence		
(0 , 1)	(1 , 2)	(2 , 3)	(3 , 4)
(5 , 6)	(4 , 5)	(6 , 7)	(7 , 8)
(10 , 11)	(8 , 9)	(9 , 10)	(11 , 12)
...	(12 , 13)	(13 , 14)	...

Here are the relations at the concrete level:

$$\forall (a, a') \in \mathbb{N}^2, a \approx_1 a' \overset{d}{\iff}\\ (q_1 = q_2) \wedge ((r_1, r_2) \in [0, 1]^2 \vee (r_1 = r_2 \wedge r_1 \notin [0, 1]))\\ \forall (a, a') \in \mathbb{N}^2, a <_1 a' \overset{d}{\iff}\\ (q_1 < q_2) \vee ((q_1 = q_2) \wedge (r_1 < r_2) \wedge (r_2 \neq 1))$$

Our example exhibits two different couples of relations, which should be in a situation of refinement.

4.3 Our proposal

As an attempt to formalize this approach, we propose to connect different levels of abstraction through the strict partial order they carry. We define the following relation $<_r$ to ensure the underlying relations fulfil the right conditions to maintain the integrity of the different representations regarding the semantics of refinement:

$$
\begin{array}{l}
\forall I \in \Omega, \forall (<_c, <_a, \approx_c, \approx_a) \in (I \times I)^4 : \\
(<_c, \approx_c) <_r (<_a, \approx_a) \overset{d}{\iff} \forall (i_1, i_2) \in I : \\
\qquad i_1 <_c i_2 \Rightarrow i_1 <_a i_2 \vee i_1 \approx_a i_2 \qquad (1) \\
\qquad \wedge \quad i_1 <_a i_2 \Rightarrow i_1 <_c i_2 \qquad (2) \\
\qquad \wedge \quad i_1 \approx_c i_2 \Rightarrow i_1 \approx_a i_2 \qquad (3) \\
\qquad \wedge \quad i_1 \approx_a i_2 \Rightarrow i_1 \approx_c i_2 \vee i_1 <_c i_2 \vee i_2 <_c i_1 \qquad (4)
\end{array}
$$

$<_r$ is defined as a relation between pairs of relations ($<_c$,$\approx_c$) and ($<_a$,$\approx_a$) that represent the strict precedence and the equivalence composing the strict partial orders bound to both levels of abstraction. In this definition, the level annotated by the index c is the more concrete level and a is the more abstract. We state what it means for a pair of relations to refine another pair of relation. These relations are defined on the set I of instants. We can only compare pairs of relations that are bounded to the same set. This definition is composed of four predicates, each of which indicate how one of the four relations is translated into the other level of observation:

1. If a strictly precedes b in the lower level, then it can either be equivalent to it in the higher level or still precede it.

2. However, if a strictly precedes b in the higher level, then it can only still precede it in the lower level. This direction doesn't allow any loss of information.

3. On the contrary, if a is equivalent to b in the lower level, it can only stay equivalent in the higher level.

4. If a is equivalent to b in the higher level then we only assure that these two instants are still related in the lower level. We can gain information this way.

This definition can be extended to strict partial orders: A strict partial order refines another when their underlying relations are in a relation of refinement.

5. RELATED WORKS

Our work takes place in GEMOC that mixes both horizontal and vertical separation of concerns. Indeed, GEMOC allows to define the various DSMLs used to model the various parts in a CPS in the various phases of the development. GEMOC relies on the Clock Constraint Specific Language (CCSL) in order to model both the MoC for the various DSML [6, 8, 12] and the coordination between DSML using the BEhavioural COOrdination Language (BeCooL) [11].

Our approach is motivated by the lack (to our knowledge) of formal definition of behavioural refinement in a context of denotational semantics. Refinement has already been studied widely through operational semantics [15, 13], and is the core concept advocated in developing correct–by–construction systems with the B [1] and Event-B [2] methods.

Our proposal provides a mechanized relation of refinement in Agda and aims at being coupled to a previous work on a mechanisation of the semantics of CCSL in the same proof assistant, based on a paper denotational semantics [7]. Thus, this approach could be reused for any other concurrent languages. Formal mechanization of temporal languages has already been done using other formal methods, for example [10] uses Higher Order Logic in Isabelle/HOL; [9] and [14] use the Calculus of Inductive Constructions in Coq, see [4].

6. CONCLUSION

This paper presented a mathematical relation over strict partial order whose goal is to model behavioural refinement in a denotational manner. Each level of abstraction is associated to a specific strict partial order while our relation binds them together. This definition has been mechanized in the Agda proof assistant, which allowed us to prove several properties about it as well as connect it to the mechanization of CCSL we made in a previous work. The bridge between these contributions has allowed us to prove the preservation of several CCSL operators through our relation of refinement. This work will lead to an extension of CCSL with a refinement operator which will allow both comparing language semantics in GEMOC and conducting correct–by–construction developments more easily with CCSL. The whole development, including the parts about CCSL, is available online on the first author's web page.

7. REFERENCES

[1] J. Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[2] J. Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[3] R. Back and J. von Wright. Trace refinement of action systems. In *CONCUR '94, Concurrency Theory, 5th Intl. Conf., Uppsala, Sweden, Aug. 22-25, Proc.*, pages 367–384, 1994.

[4] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. 2004.

[5] A. Cavalcanti and M. Gaudel. A note on traces refinement and the *conf* relation in the unifying theories of programming. In *Unifying Theories of Programming, 2nd Intl. Symp., UTP 2008, Dublin, Ireland, Sep. 8-10, Revised Selected Papers*, pages 42–61, 2008.

[6] B. Combemale, J. DeAntoni, M. V. Larsen, F. Mallet, O. Barais, B. Baudry, and R. B. France. Reifying concurrency for executable metamodeling. In *Software Language Engineering - 6th Intl. Conf., SLE 2013, Indianapolis, IN, USA, Oct. 26-28. Proc.*, 2013.

[7] J. Deantoni, C. André, and R. Gascon. CCSL denotational semantics. Research Report RR-8628, 2014.

[8] J. DeAntoni, P. I. Diallo, C. Teodorov, J. Champeau, and B. Combemale. Towards a meta-language for the concurrency concern in dsls. In *Proc. of the 2015 Design, Automation & Test in Europe Conf. & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, 2015.

[9] M. Garnacho, J. Bodeveix, and M. Filali-Amine. A mechanized semantic framework for real-time systems. In *Formal Modeling and Analysis of Timed Systems - 11th Intl. Conf., FORMATS 2013, Buenos Aires, Argentina, August 29-31, 2013. Proc.*, 2013.

[10] R. Hale, R. Cardell-Oliver, and J. Herbert. An embedding of timed transition systems in HOL. *Formal Methods in System Design*, 3(1/2), 1993.

[11] M. E. V. Larsen, J. DeAntoni, B. Combemale, and F. Mallet. A behavioral coordination operator language (bcool). In *18th ACM/IEEE Intl. Conf. on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, Sep. 30 - Oct. 2.*, 2015.

[12] F. Latombe, X. Crégut, B. Combemale, J. DeAntoni, and M. Pantel. Weaving concurrency in executable domain-specific modeling languages. In *Proc. of the ACM SIGPLAN Intl. Conf. on Software Language Engineering, SLE 2015, Pittsburgh, PA, USA, Oct. 25-27*, 2015.

[13] D. Murphy and D. Pitt. *Real-timed concurrent refineable behaviours*, pages 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[14] C. Paulin-Mohring. Modelisation of timed automata in coq. In *Theoretical Aspects of Computer Software, 4th Intl. Symp., TACS 2001, Sendai, Japan, October 29-31, 2001, Proc.*, 2001.

[15] G. Ramanathan. Refinement of events in the development of real-time distributed systems. *Theoretical Computer Science*, 133(2):341 – 359, 1994.