Generic Traces and Constraints, GenTra4CP revisited

Pierre Deransart

Generic Traces and Constraints, GenTra4CP revisited

Pierre Deransart

Inria Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France
Pierre.Deransart@inria.fr

Abstract. The generic trace format GenTra4CP has been defined in 2004 with the goal of becoming a standard trace format for the observation of constraint solvers over finite domains. It has not been used since. This paper defines the concept of generic trace formally, based on simple transformations of traces. It then analyzes, and occasionally corrects, shortcomings of the proposed initial format and shows the interest that a generic tracer may bring to develop portable applications or to standardization efforts, in particular in the field of constraints.

Keywords: Generic Trace, Constraints, Observational Semantics, Formal Specification, Portability, Standardization

1 Introduction

Following the RNTL OADymPPaC project [9], a generic trace format, called GenTra4CP (Generic Trace for CP), has been proposed in 2004 in order to specify traces of CSP(FD) resolution. One of the objective was to allow the development of portable powerful tools for solvers analysis. This format was designed as a kind of standard, consisting of a precise syntax of trace events including an XML DTD, and an operational semantics, called observational semantics, which is a partial operational semantics applicable to a set of finite domains solvers.

Such "standard" conforming tracers were implemented in four solvers. Several tools for analysis of resolution and search strategies were developed in four different environments, just using the generic trace GenTra4CP. They have been used with success after a minimal customization work for each of them. However, at that time, no formal characterization of the generic nature of the trace format has been given. Even if the implementation of the tools starting from a well defined generic trace could be realized without difficulty, and even if there was obtained a considerable gain in portability, it was virtually impossible to assess in advance the effort of adaptation needed for a solver to use the tools. Moreover it was not always possible to figure out exactly what some tool was actually observing. GenTra4CP format has been used only in the project in which it has been defined.

This paper attempts to overcome these limits, by defining formally the concept of generic trace. It analyzes formally the nature of the generic format GenTra4CP and its limitations that could have made difficult its broader use. It also shows the interest that a generic tracer approach may bring for standardization efforts and portability of applications, in particular for constraints, in proposing an approach of trace based semantics grounded on a partial operational semantics.

After an introductory section on operational semantics of traces, the Section 3 introduces some simple relations between traces in order to formalize the concept of generic trace, and to provide a proof method of compliance of a particular process trace with the generic one. The Section 4 explains the generic approach and its interest for portability. The Section 5 applies this approach to the case of GenTra4CP verifying the compliance of a particular solver. The formal description of the GenTra4CP trace format is borrowed from [9] and from [6] for the solver. This allows a better understanding of the strengths and limitations of this approach as introduced in 2004. We can then establish a possible link between the efforts of constraints standardization, and the specification method based on generic trace.

2 Preliminaries

A *trace* object consists of an initial state s_0 followed by an ordered finite or infinite sequence of *trace events*, denoted $< s_0, \overline{e} >$. $\mathcal{T}$ is a set of traces. A *prefix* (finite, of size t) of a trace $T = < s_0, \overline{e_n} >$ (finite or infinite, here of size $n \geq t$) is a partial trace $U_t =< s_0, \overline{e_t} >$ which corresponds to the t first events of T, with an initial state at the beginning. A prefix consisting of just an initial state is of size 0. The set of all the prefixes of $\mathcal{T}$ is denoted $Pref(\mathcal{T})$, with $\mathcal{T} \subseteq Pref(\mathcal{T})$.

Every trace can be decomposed into segments containing trace events only, except prefixes which start with a state. An associative operator of concatenation may be used to denote sequences concatenations. The neutral element is ϵ (empty sequence). A trace may describe state transitions, such that a segment may also be represented as sT_ts_t where s is the state in which the sequence T_t starts and s_t the state reached after the last trace event in the sequence. A segment (or prefix) of size 0 is either an empty sequence or a state.

A *domain of traces* over $\mathcal{T}$, $\mathcal{DT}_\mathcal{T}$, is a set whose elements are sets of all prefixes of one or more traces of $\mathcal{T}$. An element is prefix closed. Such a set is closed by union and intersection, and, two included elements are such that the smaller contains all the prefixes of some traces of the largest. A trace domain is a complete lattice denoted $\mathcal{DT}_\mathcal{T}(\subseteq, \bot, \top, \cup, \cap)$ where $\bot$ is the empty set and $\top = Pref(\mathcal{T})$.

Traces are used to represent the evolution of systems by describing the evolution of their state. We will distinguish two kinds of traces:

- the *virtual traces* ($\mathcal{T}^v$) whose events have the form $e = (r, s)$ where r is a *type of action* associated with a state transition and s, called *virtual state*,

the new state reached by the transition and described by a set of *parameters*. Virtual trace corresponds to sequences of states of an observed system.
- the *actual traces* ($\mathcal{T}^w$) whose events have the form $e = (a)$ where a is an *actual state* described by a set of *attributes*. Actual traces corresponds to sequences of events produced by a tracer of an observed system. Thy usually encode states changes in a synthetic manner.

We give here a simplified but sufficient definition of observational semantics. More general definitions can be found in [2].

Definition 1 (Observational Semantics).
An observational semantics consists of $< S, R, A, T, E, I, S_0 >$*, where*

- S*:* domain of virtual states*,*
- R*: finite set of* action types*, set of identifiers labeling the transitions.*
- A*:* domain of actual states*,*
- T*:* state transition function $T : R \times S \rightarrow S$*, denoted* $T(r, s) = s'$ *or* $T(r, s, s')$ *if it is a relation,*
- E_l*:* local trace extraction function $E_l : S \times R \times S \rightarrow A$*,*
- I_l*:* local trace reconstruction function $I_l : S \times A \rightarrow R \times S$*,*
- $S_0 \subseteq S$*, set of* initial states*.*

The extraction and reconstruction functions can be extended into functions E (resp. I) between sets of virtual and actual traces, and must verify the relation of *faithfulness*, $I = E^{-1}$. Local and extended functions satisfy the properties:

$E(s_0 e_1 ... e_i ...) = s_0 E_l(s_0, r_1, s_1) ... E_l(s_{i-1}, r_i, s_i) ...$ with $E_l(s_{i-1}, r_i, s_i) = a_i$, and

$I(s_0 a_1 ... a_i ...) = s_0 I_l(s_0, a1) ... I_l(s_i, a_{i+1}) ...$ with $I_l(s_{i-1}, a_i) = (r_i, s_i)$.

The local and transition functions may be represented by rules as illustrated by the Figure 1.

The observational semantics of an observed process can be considered as an abstraction of some refined operational semantics [1]. This relation will be expressed here as a relation between domains of traces. Such a relation may be expressed either between virtual or actual traces. Due to the faithfulness property, the abstraction function D_w on actual traces verifies with D_v, the abstraction function on virtual traces, the following relations: $D_v = E_c \circ D_w \circ I_d$ and $D_w = I_c \circ D_v \circ E_d$.

In the following it will be assumed that the faithfulness property is satisfied, whatever is the abstraction level of the trace description. In this case, the extraction function is deducible from the reconstruction one and reciprocally. Therefore it is sufficient to specify the transition function with the extraction only or with the reconstruction. In practice, only actual traces are manipulated by the users, but thanks to the faithfulness property, for validation purposes, the virtual trace may be used.

$$\textsf{reduce}\ \frac{< \mathcal{D}(v),\quad S_e,\quad A >}{< \mathcal{D}(v) - \Delta_v^c,\quad S_e \cup \bar{a},\quad A' >}\ \left\{\begin{array}{l}\text{remove } \Delta_v^c),\quad a \text{ wake up } c\\ (c,a) \in A,\quad A' = A - \{(c,a)\}\\ v \in \mathbf{var}(c),\quad \text{generate } \bar{a}\end{array}\right\}$$

$$\textsf{reduce}\ \frac{< \mathcal{D}(v),\quad S_e,\quad A' \cup \{(c,a)\} > \rightarrow < \mathcal{D}'(v),\quad S'_e,\quad A' >}{[\textsf{reduce},\quad c,\quad v,\quad (S'_e - S_e),\quad (\mathcal{D}(v) - \mathcal{D}'(v)),\quad a]}\ \{\}$$

$$\textsf{reduce}\ \frac{[\textsf{reduce},\quad c,\quad v,\quad \bar{a},\quad \Delta_v^c,\quad a]}{< \mathcal{D}(v),\quad S_e,\quad A > \rightarrow < \mathcal{D}(v) - \Delta_v^c,\quad S_e \cup \bar{a},\quad A - (c,a) >}\ \{\}$$

Fig. 1. Example of description of reduce in the OS of GenTra4CP (Section 5) with transition rule, extraction and reconstruction. Computations are specified on the right side

3 Abstraction relations: subtraces and derivations

We introduce simple transformations on traces: subtraces and derivations. As it is sufficient to describe transformations on the virtual traces, they are described using one part of their description, namely $< S, R, T, S_0 >$ only.

Subtraces are obtained by considering a subset of parameters.

Definition 2 (Subtrace).

Given a set of virtual traces $\mathcal{T}$ defined by $< S, R, T, S_0 >$, if $S' \subseteq S$ is defined on a subset of parameters which do not depend[1] *on any other parameter of $S - S'$, $R' \subseteq R$ is a subset of action types which use or modify these parameters only such that no other action type of $R - R'$ modifies them, S'_0 is the restriction of S_0 to S', and T' the restriction of T to S' and R', then the set of traces $\mathcal{T}'$ defined by $< S', R', T', S'_0 >$ is a (parametric) subtrace of $\mathcal{T}$, denoted $Sub_P(\mathcal{T}, \mathcal{T}')$.*

Note: it is possible that $S' \subseteq S$ and $R' = R$ ($S - S'$ contains redundant parameters, i.e. which depend only on the other parameters and thus may be removed).

Definition 3. (Derivation field and derived trace)

Given two sets of traces $\mathcal{T}_c$ and $\mathcal{T}_d$, where $\mathcal{T}_c$ and $\mathcal{T}_d$ are said respectively concrete *and* derived, *$\mathcal{T}_d$ is a derivation field of $\mathcal{T}_c$ by D if there exists a mapping $D : Pref(\mathcal{T}_c) \rightarrow Pref(\mathcal{T}_d)$, called a* derivation, *such that for all finite derived prefixes t_d of size n and for all concrete prefix t_c such that $D(t_c) = t_d$, there exists an increasing chain of concrete prefixes $[t_c^0, t_c^1, ..., t_c^i, ..., t_c^{n-1}, t_c]$ (not necessarily contiguous), such that*

- $D(t_c^0) \in S_{0,d}$,
- $\forall i > 0$ *if* $D(t_c^i) = t_d^i$ *with* t_d^i *prefix of* t_d *made of the* i *first events, then* $D(t_c^{i+1}) = t_d^{i+1}$.

If D is surjective, the set $\mathcal{T}_d$ is called derived trace *by D of $\mathcal{T}_c$, noted $Drv_D(\mathcal{T}_c, \mathcal{T}_d)$.*

[1] A parameter p depends on p' iff p' is used in the computation of p in some transition.

As defined, D is a partial function. It can be made total by considering that all elements of $S_{0,c}$ have an image in $S_{0,d}$ and that the image of each prefix between t_c^i and t_c^{i+1} is $D(t_c^i)$.

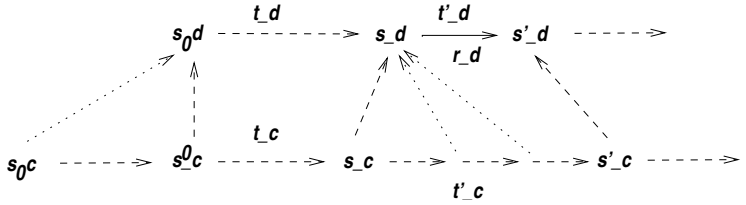

Fig. 2. Derivation (dashed arrow correspond to the totalized derivation), t_x denotes a prefix

This approach puts emphasis on traces without considering the way they have been produced or the way they are specified. The main idea is that a trace transformation is the result of a computation on likely full prefixes of concrete traces, represented by de derivation D.

Property 1.

Given two derivations D_1 and D_2, if D_1 is surjective or if D_2 is total, $D_1 \circ D_2$ is a derivation.

A parametric subtrace (definition 2) is a derived trace.

The following establishes a method of proof that two sets of traces specified by transition relations are related by a derivation.

Definition 4. (Simulable Trace)

Given two sets of traces $\mathcal{T}_c$ (concrete) and $\mathcal{T}_d$ (derived), respectively defined with $< S_c, R_c, T_c, S_{0,c} >$ and $< S_d, R_d, T_d, S_{0,d} >$, $\mathcal{T}_c$ is simulable *in $\mathcal{T}_d$ if R_c and R_d are in a one-one mapping h, and if there exists an application $d : S_c \to S_d$ such that:*

- $\forall s_0 \in S_{0,c}, d(s_0) \in S_{0,d}$.
- $\forall r_c \in R_c, s_c, s'_c \in S_c,\ T_c(s_c, r_c, s'_c) \Rightarrow \exists s_d, s'_d \in S_d, d(s_c) = s_d \wedge d(s'_c) = s'_d \wedge T_d(s_d, h(r_c), s'_d)$.

Theorem 1.

Given two sets of traces $\mathcal{T}_c$ (concrete) and $\mathcal{T}_d$ (derived), such that $\mathcal{T}_c$ is simulable in $\mathcal{T}_d$, then $\mathcal{T}_d$ is a derivation field for $\mathcal{T}_c$ and the corresponding derivation is total.

Corollary 1.

Given two sets of traces $\mathcal{T}$ and $\mathcal{T}'$ such that there exists a parametric subtrace of $\mathcal{T}$ simulable in $\mathcal{T}'$, then $\mathcal{T}'$ is a derivation field for $\mathcal{T}$.

4 Generic Trace

The idea of generic trace meets the needs of specification and portability. It is intended to specify a process or an algorithm by its observable behavior, i.e. the trace of abstracted operations that it is expected to implement. The level of description must be general enough to include family of processes, and the level of granularity must be sufficiently refined to be used by a family of applications. This may be the case for example for applications such as monitoring, debugging, visualization tools, or any application using the generic trace.

Definition 5 (Generic Trace (GT)).
Given a family of processes $p \in P$, each of them equipped to produce traces $\mathcal{T}_p$, a set of traces $\mathcal{T}_g$ is generic if, for each process p in the family, there exists a derivation D_p of its traces which is a parametric subtrace of $\mathcal{T}_g$, that is:
$\forall p \in P,\ \exists\ \mathcal{T}$ *such that* $Drv_{D_p}(\mathcal{T}_p, \mathcal{T}) \wedge Sub_P(\mathcal{T}_g, \mathcal{T})$.

Three questions are then worth posing:

- How to ensure that the trace produced by some process is compliant with the GT?
- Can the GT be used in application development, with the guarantee that the application will work with any compliant process?
- Can the GT be extended to handle more processes in such a way that existing applications will still work?

Here are some possible answers.

Compliance to the Generic Trace

A trace of a process is compliant w.r.t. the GT if it satisfies the definition 5, i.e. there exists a subtrace of the GT which is a derivation of a subtrace of those of the process. It is thus possible either to implement straightforwardly the GT as it is (in this case the process produces exactly the GT), or to prove that the traces a process p may generate verifies $\exists \mathcal{T}', Drv_{D_p}(\mathcal{T}_p, \mathcal{T}') \wedge Sub_p(\mathcal{T}_g, \mathcal{T}')$.

Building tools with the Generic Trace

The interest of a generic trace is that it facilitates the development of tools that can be used with all compliant processes. The development is made considering that the tool uses at least a sub-GT covering sufficiently many processes. Thus it is possible to adapt the tool to the process p by applying to the trace generated by the process (without any modification) the derivation D_p to get a GT. This can be done at le level of the process (process can use any tool) or at the level of the tool (tool can be run with this particular process). The Figure 3 illustrates these two ways to adapt processes with compliant tracer and tools.

The fact that the GT has a formal specification makes it possible to realize a prototype (executable specification) which shall be itself a new compliant process. It is thus possible to use such a prototype to develop and test tools. This development method guarantees that any tool made on the top of the GT will be able to work with any compliant processes.

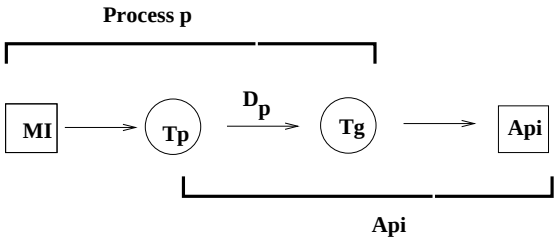

Fig. 3. Use of a Generic Trace: process or application adaptation

Generic Trace extensions

As long as an extension of the GT preserves the fact that a process is compliant w.r.t. a subtrace of the extended GT, they still are compliant w.r.t. the extended GT. It is sufficient to ensure that any GT extension preserves the parametric subtraces. This guarantees that the compliant processes will continue to be usable by tools using the original GT.

5 The Generic Trace GenTra4CP

In the final document [9], the generic trace GenTra4CP is defined with an observations semantics whose transition function is defined with a subset of parameters. Thus only a generic subtrace has a formal semantics. The other parameters are described informally by the description of other attributes of the actual trace. Their syntax is fixed by a DTD XML and informal explanations are provided for each new attribute. We recall here the semantics as originally presented in [9] (section 3.3.1)[2]

`Beginning of Citation`:

Definition 6. *(Solver State)*

A solver state is a 8-tuple: $\mathbb{S} = (\mathcal{V}, \mathcal{C}, \mathcal{D}, A, E, R, S_c, S_e)$
where: $\mathcal{V}$ *is the set of declared variables;* $\mathcal{C}$ *is the set of declared constraints;* $\mathcal{D}$ *is the function that assigns to each variable in* $\mathcal{V}$ *its domain (a set of values in the finite set* D*);* A *is the set of active pairs of the form (constraint, solver event*[3]*);* E *is the set of solved constraints;* R *is the set of unsatisfiable (or rejected) constraints.* S_c *is the set of sleeping constraints;* S_e *is the set of solver events to propagate ("sleeping events").*

[2] Here one uses n instead of ν to denote the current node of the choice-tree.

[3] This work inherits from two areas, constraint solving and debugging, which both use the word "event" in correlated but different meanings: a *solver event* is produced by the solver and has to be propagated (e.g. the update of the domain bounds of a variable); a *trace event* corresponds to an execution step which is worth reporting about.

Control		Propagation	
new variable	$v,\quad D_{v,i}$	reduce	$c,\quad v,\quad \bar{a},\quad \Delta_v^c,\quad \mathrm{a}$
new constraint	c		
post, remove	c	suspend, solved	c
restore	$v,\quad \Delta_v$	reject	$c,\quad a$
choice point	n	awake	$c,\quad a$
back to	$n,\quad n'$	schedule	$c,\quad a$
solution, failure	n		

Table 1. Attributes of the actual trace of GenTra4CP

A, S_c, E and R are used to describe four specific states of a constraint during the propagation stage: active, sleeping, solved or rejected.

The *store of constraints* is the set of all constraints taken into account. The store is called σ in the following and defined as the partition $\sigma = \{c \mid \exists (c, a) \in A\} \cup S_c \cup E \cup R$. All the constraints in σ are defined, thus $\sigma \subseteq \mathcal{C}$. The set of variables involved in the constraint c is denoted by $\mathbf{var}(c)$. The predicate $false(c, \mathcal{D})$ (resp. $solved(c, \mathcal{D})$) holds when the constraint c is considered as unsatisfiable (resp. solved: it is universally true and does not influence further reductions any more) by the domains in $\mathcal{D}$.

The search is often described as the construction of a search-tree.

Definition 7. *(Search-Tree State)*
The search-tree is formalized by a set of ordered labeled nodes $\mathcal{N}$ representing a tree, and a function Σ which assigns to each node a solver state. The nodes in $\mathcal{N}$ are ordered by the construction. Three kinds of nodes are defined and characterized by three predicates: failure leave ($failed(\mathbb{S})$), solution leave ($solution(\mathbb{S})$), and choice-point node ($choice\text{-}point(\mathbb{S})$). The last visited node is called current node *and is denoted n. The usual notion of* depth *is associated to the search-tree: the depth is increased by one between a node and its children. The function δ assigns to a node n its depth $\delta(n)$.Therefore, the state of the search-tree is a quadruple: $\mathbb{T} = (\mathcal{N}, \Sigma, \delta, n)$.*

In the initial solver state, n_0 denotes the root of the search-tree and all the sets that are part of $\mathbb{S}$ are empty.

`End of Citation`

The remaining description consists of the description of each event type of the actual trace (called in [6] "generic trace schema") by introducing other attributes (some of them are redundant like external and internal constraint identifier). One illustrates the methodology of GT construction by analyzing one "implementation" of the GT as presented in [6]. In this paper three "specializations" of the GT are detailed for three solvers (GNU-Prolog, Choco and PaLM). They consist of a description of the operational semantics of each solver by their transition function. We show here that the proposed OS for PaLM [5] is compliant. Among

$$\text{new variable}\ \frac{< \mathcal{V},\ \ \mathcal{D} >}{< \mathcal{V} \cup \{v\},\ \ \mathcal{D} \cup \{(v, \mathcal{D}_{v,i})\} >}\ \left\{ \begin{array}{l} v \notin \mathcal{V}, \\ \mathcal{D}(v) = D_{v,i} \end{array} \right\}$$

$$\text{new constraint}\ \frac{< \mathcal{C} >}{< \mathcal{C} \cup \{c\} >}\ \left\{ \begin{array}{l} c \notin \mathcal{C}, \\ \mathbf{var}(c) \subseteq \mathcal{V} \end{array} \right\}$$

$$\text{post}\ \frac{< A >}{< A \cup \{(c, \bot)\} >}\ \left\{ \begin{array}{l} c \in \mathcal{C}, \\ c \notin \sigma \end{array} \right\}$$

$$\text{choice point}\ \frac{< \mathcal{N},\ \ \Sigma,\ \ \mathbb{S} >}{< \mathcal{N} \cup \{n\},\ \ \Sigma \cup \{(n, \mathbb{S})\},\ \ n >}\ \left\{ \begin{array}{l} \textit{ch-pt}(\mathbb{S}), \\ n \notin \mathcal{N} \end{array} \right\}$$

$$\text{back to}\ \frac{< \mathbb{S},\ \ \nu >}{< \Sigma(n),\ \ n >}\ \left\{ \begin{array}{l} n \neq \nu,\ \ n \in \mathcal{N}, \\ \textit{ch-pt}(\mathbb{S}) \end{array} \right\}$$

$$\text{solution}\ \frac{< \mathcal{N},\ \ \Sigma,\ \ \mathbb{S} >}{< \mathcal{N} \cup \{n\},\ \ \Sigma \cup \{(n, \mathbb{S})\},\ \ n >}\ \left\{ \begin{array}{l} \textit{sol}(\mathbb{S}), \\ n \notin \mathcal{N} \end{array} \right\}$$

$$\text{failure}\ \frac{< \mathcal{N},\ \ \Sigma,\ \ \mathbb{S} >}{< \mathcal{N} \cup \{n\},\ \ \Sigma \cup \{(n, \mathbb{S})\},\ \ n >}\ \left\{ \begin{array}{l} \textit{flr}(\mathbb{S}), \\ n \notin \mathcal{N} \end{array} \right\}$$

$$\text{remove}\ \frac{< \sigma >}{< \sigma - \{c\} >}\ \{\, c \in \sigma \,\}$$

$$\text{restore}\ \frac{< \mathcal{D}(v) >}{< \mathcal{D}(v) \cup \Delta_v >}\ \left\{ \begin{array}{l} v \in \mathcal{V}, \\ \Delta_v \cap \mathcal{D}(v) = \emptyset,\ \ \Delta_v \subseteq \mathcal{D}_{v,i} \end{array} \right\}$$

Fig. 4. OS of GenTra4CP (control, without the parameter δ)

the three experimented solvers, PaLM has a clearly different semantics. The transition part of the OS is depicted in the figures 4 and 5.

In order to show that the OS (trace semantics) of PaLM is compliant, one need the following properties of the GT:

(G1) $sol(\mathbb{S}) \Rightarrow R = \emptyset$

(G2) $flr(\mathbb{S}) \Leftrightarrow R \neq \emptyset$

(G3) $(evtype = \mathsf{reduce}) \Rightarrow R = \emptyset$

(G4) $(evtype = \mathsf{awake}) \Rightarrow (R = \emptyset \wedge A = \emptyset)$

(G5) $(evtype = \mathsf{schedule}) \Rightarrow (R = \emptyset \wedge A = \emptyset)$

One admits:

(P1) $dependence(c, a) \Leftrightarrow awcond(c, a)$

(P2) $select(a) \Rightarrow \exists c \in \mathcal{C}\ action(c, a)$

(P3) $\exists v \in \mathbf{var}(()c), \mathcal{D}(v) = \emptyset \Rightarrow false(c, \mathcal{D})$

Theorem 2.

The GT restricted to all events depicted in the Figures 4 and 5 but back to *and* solved*, is a parametric subtrace of GenTra4CP, derived from the trace specified for PaLM (Figures 6 and 7).*

$$\text{reduce}\ \frac{< \mathcal{D}(v),\ \ S_e,\ \ A >}{< \mathcal{D}'(v),\ \ S'_e,\ \ A' >}\ \left\{\begin{array}{l}\mathcal{D}'(v) = \mathcal{D}'(v) - \Delta^c_v,\ \ \text{supprime } \Delta^c_v,\\ (c,a) \in A,\ \ v \in \mathbf{var}(c),\ \ \text{Red. gn. } \bar{a},\\ A' = A - (c,a),\ \ S'_e = S_e \cup \bar{a}\end{array}\right\}$$

$$\text{suspend}\ \frac{< A,\ \ S_c >}{< A - \{(c,a)\},\ \ S_c \cup \{c\} >}\ \{(c,a) \in A\}$$

$$\text{solved}\ \frac{< A,\ \ E >}{< A - \{(c,a)\},\ \ E \cup \{c\} >}\ \left\{\begin{array}{l}(c,a) \in A,\\ solved(c,\mathcal{D})\end{array}\right\}$$

$$\text{reject}\ \frac{< A,\ \ R >}{< A - \{(c,a)\},\ \ R \cup \{c\} >}\ \left\{\begin{array}{l}(c,a) \in A,\\ false(c,\mathcal{D})\end{array}\right\}$$

$$\text{awake}\ \frac{< A,\ \ S_c >}{< A \cup \{(c,a)\},\ \ S_c - \{c\} >}\ \left\{\begin{array}{l}c \in S_c,\ \ a \in S_e \cup \{\bot\},\\ awcond(c,a)\end{array}\right\}$$

$$\text{schedule}\ \frac{< S_c,\ \ S_e >}{< S'_c,\ \ S'_e >}\ \left\{\begin{array}{l}c \in S_c,\ \ e \in S_e,\\ action(c,a)\end{array}\right\}$$

Fig. 5. OS of GenTra4CP (propagation)

new variable, new constraint idem GenTra4CP

post , choice point idem GenTra4CP

$$\text{solution}\ \frac{< \mathcal{N},\ \ \Sigma,\ \ \mathbb{S} >}{< \mathcal{N} \cup \{n\},\ \ \Sigma \cup \{(n,\mathbb{S})\},\ \ n >}\ \left\{\begin{array}{l}sol(\mathbb{S}),\\ n \notin \mathcal{N}\end{array}\right\}$$

$$\text{failure}\ \frac{< \mathcal{N},\ \ \Sigma >}{< \mathcal{N} \cup \{n\},\ \ \Sigma \cup \{(n,\mathbb{S})\},\ \ n >}\ \left\{\begin{array}{l}n \notin \mathcal{N},\\ R \neq \emptyset\end{array}\right\}$$

remove idem GenTra4CP

$$\text{restore}\ \frac{< \mathcal{D}(v),\ \ Q_t,\ \ \mathcal{E} >}{< \mathcal{D}(v) \cup R_v,\ \ Q_t \cup \bar{a},\ \ \mathcal{E} - E >}\ \left\{\begin{array}{l}v \in \mathcal{V},\ \ R_v \subseteq \{d \in \mathbb{D} | \mathcal{E}(v,d) \cap \sigma \neq \emptyset\},\\ E = \{\mathcal{E}(v,d) | d \in R_v\},\\ \bar{a} \text{ actions de restauration de } \mathcal{D}(v)\end{array}\right\}$$

Fig. 6. OS of PaLM [6] (control)

6 Generic Trace and Constraints Specification

This approach of semantics can be applied to constraints specification. The question then is whether it exists a generic trace covering all the constraints that one wishes to describe, i.e. covering different types of constraints (single, global, ...), different domains (FD, intervals, ...), different classes of solvers (CSP, SAT, rules, such as CHR), different levels (algorithms, modules, modeling) or different aspects (language, interaction, interfaces, ...) as well.

We limit ourselves here to the CSP case. Each constraint has a declarative semantics defined by the relation it represents on its domains. The GT can thus provide a description of the possible effects of each constraint separately or in a network, regardless the particular algorithm it implements. In this sense such semantics is a kind of minimal description of what we should be able to observe

$$\textsf{reduce}\ \frac{< \mathcal{D}(v),\ \ Q_t,\ \ \mathcal{E} >}{\left\{ \begin{array}{l} < \mathcal{D}(v) - \Delta_v^{c_a},\ \ Q_t \cup \{\bar{a}\}, \\ \mathcal{E} \cup \{(v,d,C) \mid d \in \Delta_v^{c_a}\} > \end{array} \right\}} \left\{ \begin{array}{l} v \in \mathbf{var}(c),\ \ R = \emptyset,\ \ A = \{(c,a)\}, \\ \Delta_v^{c_a} \neq \emptyset \text{ set of inconsistent values for } v, \\ C \subseteq \sigma \text{ explains the removal of } \Delta_v^{c_a} \text{ from } \mathcal{D}(v), \\ \text{The reduction generates } \bar{a} \end{array} \right\}$$

$$\textsf{suspend}\ \frac{< A,\ \ S_c >}{< \emptyset,\ \ S_c \cup \{c\} >}\ \{A = \{(c,a)\}\}$$

$$\textsf{reject}\ \frac{< A,\ \ R >}{< \emptyset,\ \ R \cup \{c\} >}\ \left\{ A = \{(c,a)\},\ \ v \in \mathbf{var}(c),\ \ \mathcal{D}(v) = \emptyset \right\}$$

$$\textsf{awake}\ \frac{< S_c,\ \ A >}{< S_c - \{c\},\ \ \{(c,a)\} >}\ \left\{ \begin{array}{l} A = \emptyset,\ \ c \in S_c,\ \ R = \emptyset, \\ a \in Q_h \cup \{\bot\},\ \ \mathit{dependence}(c,a) \end{array} \right\}$$

$$\textsf{schedule}\ \frac{< Q_h,\ \ Q_t >}{< \{a\},\ \ Q_t - \{a\} >}\ \left\{ \mathit{select}(a),\ \ A = \emptyset,\ \ a \in Q_t,\ \ R = \emptyset S_c \neq \emptyset \right\}$$

Fig. 7. OS of PaLM [6] (propagation)

of the behavior of a constraints set. It can be used to define any kind of interfaces, particularly for problem modeling.

In practice, as is what has been done for GenTra4CP, one should start with a definition of an actual trace whose meaning can be given by a reconstruction function. It should be completed by an OS as large as possible such that parameters relevant to potential interfaces and applications are fully described.

We illustrate this approach of a generic semantics with a simple resolution example, showing the two traces obtained with GNU-Prolog and PaLM for this example. Both solvers have been instrumented to produce the generic trace for CSP(FD), and their traces can be "understood" using the OS of the Figure 8. Both traces (Figure 9) correspond to the resolution of (GNU-Prolog syntax) `fd_element_var(I,[2,5,7],A), (A#=I ; A#=2)` which admits one solution only[4]. The declarative semantics of this constraint (all variables are finite domain) can be expressed as: `fd_element_var(I, L, V)` (`L` liste) constrains `V` to be equal to the `I`ith element of `L`. Thas is to say all triples such that $i \in$ `I`, $u \in$ `L`(i), $v \in$ `V` and $u = v$ are valid. The interval `[`*a*-*b*`]` denotes *from a to b* and `[`*a*,*b*`]`, *a and b*. One may observe[5] that the traces are different, so, in particular:

- the domain of `I` is not the same for GNU (`[1-3]`) and for PaLM (`[0-2]`);
- the order and the values of the values removal are not the same, as the choice of variables to consider;
- search spaces are different;
- a specific variable occurs in the trace of PaLM (`v-1`).

[4] PaLM produces shortcuts such that the sequence reduce suspend schedule awake is displayed as reduce awake. Such shortcut does not have any semantics in GenTra4CP (it could be adapted). This shows only that the PaLM OS given in [6] was not actually compliant to the GT.

[5] GenTra4CP produces traces in XML, readable but verbose. A more concise representation has been adopted here.

$$\textsf{new variable}\ \frac{[\textsf{new variable},\quad v,\quad D_{v,i}]}{<\mathcal{V},\quad \mathcal{D}>\rightarrow<\mathcal{V}\cup\{v\},\quad \mathcal{D}\cup\{(v,D_{v,i})\}>}\ \{\}$$

$$\textsf{new constraint}\ \frac{[\textsf{new constraint},\quad c]}{<\mathcal{C}>\rightarrow<\mathcal{C}\cup\{c\}>}\ \{\}$$

$$\textsf{post}\ \frac{[\textsf{post},\quad c]}{<A>\rightarrow<A\cup\{(c,\bot)\}>}\ \{\}$$

$$\textsf{choice point}\ \frac{[\textsf{choice point},\quad n]}{<\mathcal{N},\quad \Sigma,\quad \mathbb{S}>\rightarrow<\mathcal{N}\cup\{n\},\quad \Sigma\cup\{(n,\mathbb{S})\},\quad n>}\ \{\}$$

$$\textsf{reduce}\ \frac{[\textsf{reduce},\quad c,\quad v,\quad \bar{a},\quad \Delta_v^c,\quad a]}{<\mathcal{D}(v),\quad S_e,\quad A>\rightarrow<\mathcal{D}(v)-\Delta_v^c,\quad S_e\cup\bar{a},\quad A-(c,a)>}\ \{\}$$

$$\textsf{suspend}\ \frac{[\textsf{suspend},\quad c,\quad a]}{<A,\quad S_c>\rightarrow<A-\{(c,a)\},\quad S_c\cup\{c\}>}\ \{\}$$

$$\textsf{awake}\ \frac{[\textsf{awake},\quad c,\quad a]}{<A,\quad S_c>\rightarrow<A\cup\{(c,a)\},\quad S_c-\{c\}>}\ \{\}$$

Fig. 8. OS of GenTra4CP (reconstruction)

These variations are irrelevant when comparing the respective semantics (renaming, extra variable) and from both actual traces one may reconstruct the corresponding virtual ones. However some variations should be examined and fixed like the limit values of `I`, or some specific attributes.

7 Discussion

The semantics of traces presented here corresponds to the "Observable Semantics" of Lucas [8] or the partial trace semantics of Cousot [1]. The parameters of the virtual states are, as expressed by Lucas, "syntactic objects used to represent the conduct of operational mechanisms". The traces are abstract representations of process semantics which allow to take into account the sole details we want to consider as common to a set of processes. The choice to relate two forms of trace (virtual and actual) corresponds to the need to reconcile different pragmatic approaches: formal specification of semantics more or less abstract, and empirical manipulations of traces like in trace-based systems [10]. We established here a particular method to demonstrate compliance of a process trace with regards to a generic trace. This approach allows to establish formal relations with the trace theory [3] too.

We have shown here that the definition of the trace GenTra4CP can be well defined in such a theoretical framework, and we have characterized by relatively simple transformations (parametric subtrace, similarity and derivation) the formal linkages between the observed processes and the generic trace. This analysis revealed some insufficiencies in the formal definition of GenTra4CP as the lack of formal verification of particular traces solver compliance. Simonis & al [11]

```
 1[0]choice point node(0)
 2[1]newVariable v1 [0-mx]
 3[1]newVariable v2 [0-mx]
 4[1]newConstraint c1
fd_element([v1,[2,5,7],v2])
 5[1]post c1
 6[1]reduce c1 v1 [0,4-mx]
 7[1]reduce c1 v2
   [0-1,3-4,6,8-mx]
 8[1]suspend c1
 9[1]choice point node(1)
10[2]newConstraint c4
    x_eq_y([v2,v1])
11[2]post c4
12[2]reduce c4 v2 [5,7]
13[2]reduce c4 v1 [1,3]
14[2]suspend c4
15[2]schedule v2 dom
16[2]awake c1
17[2]reject c1
18[2]failure node(2)
...
```

```
 0[0]newVariable v0 I [0-mx]
 1[0]newVariable v1 A [0-mx]
 2[0]newConstraint c0
    element(I,[2,5,7],A)
 3[0]post c0
 4[0]suspend c0
 5[0]awake c0
 6[0]reduce c0 v0 [3-mx] max
 7[0]reduce c0 v1 [0,1] min
 8[0]reduce c0 v1 [8-mx] max
 9[0]suspend c0
10[0]newConstraint c1 eq(I,A)
11[0]post c1
12[0]suspend c1
13[0]awake c0 (v0,max)
14[0]reduce c0 v1 [2-7] empty
15[0]reject c0 empty
16[0]failure
17[0]newVariable v-1 I [0-1]
18[0]reduce c2 v-1 [0,1] empty
...
```

Fig. 9. Partial actual trace of GNU-Prolog and PaLM with the given example. The second attribute is the choice-tree depth

note that the generic trace GenTra4CP contains too many details with a too sophisticated specification. This is certainly true if the objective is just to analyze the evolution of some problem variables and some aspects of the search. In this case the need of trace information is limited and it is less work to implement directly the capture of the needed information rather than implementing a full generic trace format. But it is different if the objective is to create a generic interface between solvers and many more applications. Our study shows also that GenTra4CP probably contains too many optional details with no clear semantics, such that implementers feel free not to implement many of them, or to implement them with just specific implementation dependent semantics. A more demanding approach, but which may be more useful, could be to specify formally more attributes of the generic trace.

Moreover, as it has been observed in the Section 4, it is the task of the developer of a solver to implement a generic (sub)trace or to adapt the tools which have been developed on the basis of the generic trace. The investment to make is measured by the gap between the developed process trace and the generic trace (formally a derivation, Figure 3). It may seem easier to implement an ad-hoc trace systematically, rather than to implement once a compliant tracer, or to adapt a tool each time needed. Langevine and Ducassé have shown [7] that a generic approach could have more advantages than drawbacks, but it is similar to a standardization effort.

Such an effort can only result from the action of a large community, and not from a small group as in the case of GenTra4CP. The project of standard [4] focuses mainly on the definition of a Java interface that includes in particular the major types of variables, unary constraints, some binary and global constraints, as some strategies to search for solutions. But the question of the semantics

cannot be ignored. If the declarative semantics of simple constraints poses little problem of specification, it is not the same for the operational semantics, whose accuracy depends on potential applications developed with constraint problems. The approach presented here, based on a generic trace semantics, may be a way since it provides a framework for specifying outcomes and side effects of constraint, revealing for example constraints interactions independently from specific implementations.

8 Conclusion

GenTra4CP has been an innovative approach using a partial trace semantics to handle both problems of specifying constraint solvers (on finite domains) and of portable analysis tools. Such an effort was similar to a standardization effort, but with no effective dissemination because of its limits (small group who made it, some technical gaps and restricted to one constraints domain).

We have introduced a simple formal framework based on trace theory and abstract interpretation to explain the method of generic trace construction, and to show the potential value of this approach to specify a partial semantics of constraints resolution.

The realization of a generic trace for a significant set of simple or global constraints certainly represents a considerable amount of efforts. It seems however that such an approach could not only allow the portability of potential applications, but also contribute to the semantics of knowledge representation systems which combine several methods like constraints and rules.

References

1. Cousot, P., Cousot, R.: Systematic design of program transformation frameworks by abstract interpretation. In: Proc. of POPL 2002. pp. 178–190 (2002)
2. Deransart, P.: Towards a Trace Meta-Theory (Mar 2011), working document `http://hal.inria.fr/` (mainly in French)
3. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific Publishing, Singapore (1995)
4. Feldman, J.: JSR-331, Java Constraint Programming API. Tr, Java Community Process, Cork Constraint Computation Centre (2011), `{\tthttp://www.jcp.org}`
5. Jussien, N., Barichard, V.: The PaLM system: explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000. pp. 118–133. Singapore (Sep 2000), `{http://www.emn.fr/jussien/publications/jussien-WCP00.pdf}`
6. Langevine, L., Deransart, P., Ducassé, M.: A generic trace schema for the portability of cp(fd) debugging tools. In: Apt, K., Fages, F., Rossi, F., Szeredi, P., Vancza, J. (eds.) Recent Advances in Constraints. No. 3010 in LNAI, Springer Verlag (May 2004)
7. Langevine, L., Ducassé, M.: Design and implementation of a tracer driver: Easy and efficient dynamic analyses of constraint logic programs. Theory and Practice

of Logic Programming, Cambridge University Press 8(5-6) (Sep-Nov 2008), http://arxiv.org/abs/0804.4116
8. Lucas, S.: Observable Semantics and Dynamic Analysis of Computational Processes. Tech. Rep. LIX/RR/00/02, Laboratoire d'Informatique LIX (2000), http://users.dsic.upv.es/\verb.~.slucas
9. OADymPPaC: Tools for dynamic analysis and debugging of constraint programs, french RNTL project (2001-2004) http://contraintes.inria.fr/OADymPPaC
10. Settouti, L.S.: Modeled Trace Based Systems: Model and Languages for the Use of Traces of Interactions. Ph.D. thesis, Université Claude Bernard - Lyon I (Jan 2011)
11. Simonis, H., Davern, P., Feldman, J., Mehta, D., Quesada, L., Carlsson, M.: A Generic Visualization Platform for CP. In: Petrie, K. (ed.) Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming. St Andrews, Scotland (Sep 2010)

ANNEX: Proofs

8.1 Proof of theorem 2

One shows that a parametric subtrace of PaLM is simulable by a parametric subtrace of GenTra4CP.

One considers the GT GenTra4CP, restricted to all the events of the Figures 4 and 5 but back to and solved. Ignoring back to does not affect the search-tree construction but only its visiting strategy, and ignoring solved corresponds to removing the parameter E in the solver state. Furthermore the parameter corresponding to the explanations can be ignored, as it is not formalized in the OS of GenTra4CP.

According to the definition 2, the restriction to the subset of considered events is a parametric subtrace of GenTra4CP.

The subtrace of PaLM to be considered consists just in ignoring the explanations. This does not restrict the set of action types (definition 2).

In GenTra4CP and the considered subtrace, the control part $\mathbb{T}_g$ uses actually 4 parameters: $\mathcal{N}, \Sigma, \delta, \nu$, and the propagation part $\mathbb{S}_g$ 8 parameters: $\mathcal{V}, \mathcal{C}, \mathcal{D}, A, E, R, S_c, S_e$, in total 12 parameters.

In PaLM, the control part $\mathbb{T}_p$ uses 5 parameters: $\mathcal{N}, \Sigma, \delta, \nu, Q_t$, and the propagation part $\mathbb{S}_p$ 9 parameters: $\mathcal{V}, \mathcal{C}, \mathcal{D}, A, R, S_c, Q_h, Q_t, \mathcal{E}$; in total 13 parameters (Q_t is common). There are some differences:

- E, the subset of the "constraint store", containing the valid constraints, is irrelevant in PaLM, as no satisfiability test is realized in PaLM (no "entailment").
- The set S_e of the current events in PaLM is a queue ($S_e = Q_h \cup Q_t$) whose head Q_h (a singleton) contains the selected current event.
- The state of the PaLM solver contains an additional parameter $\mathcal{E}$, the explanation function which serves to store the what is called the "explanations". E is a partial function: $\mathcal{E} : \mathcal{V} \times \mathbb{D} \longrightarrow \mathcal{P}(\sigma)$[6] which assigns to each value removal (v, d) $(v \in \mathcal{V}, d \in \mathcal{D}(v))$ a set of non relaxed constraints which explains this removal. This partial function is updated by the events reduce and restore.
- A, in PaLM, has at most one element.

Thus one shows that the PaLM subtrace is simulable in the subtrace "PaLM" of GenTra4CP.

One uses the theorem 1. One defines the application d between the modified states $\mathbb{T}_p \times \mathbb{S}_p$ and $\mathbb{T}_g \times \mathbb{S}_g$, that is: (one omits δ which is deducible directly from $\mathcal{N}$)

$\mathcal{N}, \Sigma, \nu, \mathcal{V}, \mathcal{C}, \mathcal{D}, A, R, S_c, Q_h, Q_t$ and
$\mathcal{N}, \Sigma, \nu, \mathcal{V}, \mathcal{C}, \mathcal{D}, A, R, S_c, S_e$

[6] $\mathcal{P}(\sigma)$: powerset of the store σ (instance of the constraints which are in A, S_c and R for PaLM).

as follows: identity for the 9 first parameters of PaLM $\mathcal{N}, \Sigma, \nu, \mathcal{V}, \mathcal{C}, \mathcal{D}, A, R, S_c$, then $Q_h \cup Q_t = S_e$.

The action types have the same names ans their set is restricted to those in the Figures 6 and 7.

The initial states $\mathbb{T}_{0,p} \times \mathbb{S}_{0,p}$ and $\mathbb{T}_{0,g} \times \mathbb{S}_{0,g}$ to be considered are:
$\{rc_p\}, (rc_p, \mathbb{S}_{0,p}), rc_p, \emptyset_p, \emptyset_p, \emptyset_p, \emptyset_p, \emptyset_p, \emptyset_p, \emptyset_p, \emptyset_p$ and
$\{rc_g\}, (rc_g, \mathbb{S}_{0,p}), rc_g, \emptyset_g, \emptyset_g, \emptyset_g, \emptyset_g, \emptyset_g, \emptyset_g, \emptyset_g, \emptyset_g$

new variable, new constraint, post, choice point and remove are in correspondence as the transition rules are the same, as their modified parameters as well.

For solution and failure, it is the same provided the properties (G1) and (G2) hold.

The case of restore is more complex. But, if one ignores the explanations and take for Δ_v, R_v ($\Delta_v = R_v$) for the same variable v, the conditions associated to the event of Gentra4CP are deducible from the explanations properties (restitution of the removed values, then inexistent in the current domain of v). But one has to justify the update of S_e in the transition rule of GenTra4CP.

reduce. To $\Delta_v^{c_a}$ it corresponds Δ_v^c (set of inconsistent values) of the GT. By (G3) the properties $R = \emptyset$ correspond. Finally as $d(Q_h \cup Q_t) = S_e$, then $d(Q_h \cup Q_t \cup \bar{a}) = S_e \cup \bar{a}$.

suspend. In the corresponding initial states $(c, a) \in A$, and $A' = A - \{(c, a)\}$ in the final states.

reject. Uses (P3) for the initial states, and the final states are in correspondance.

awake. Uses (P1) and (G4).

schedule. Uses (P2) and (G5). S_c and S_e are invariants in the GT.