The duality of prime event structures and automata
Course notes for CS352

R.J. van Glabbeek
Computer Science Department, Stanford University
Stanford, CA 94305-9045, USA.
`rvg@cs.stanford.edu`

1 Prime event structures with binary conflict

Prime event structures with binary conflict as a model of concurrency were introduced by NIELSEN, PLOTKIN & WINSKEL in [NPW81].

Definition 1 A *(labeled) prime event structure with binary conflict (over an alphabet Act)* is a 4-tuple $\mathrm{E} = (E, <, \#, l)$ where

- E is a set of *events*,
- $< \subseteq E \times E$ is a partial order (the *causality relation*) satisfying the *principle of finite causes*:
$$\forall e \in E : \{d \in E \mid d < e\} \text{ is finite},$$
- $\# \subseteq E \times E$ is an irreflexive, symmetric relation (the *conflict relation*) satisfying the *principle of conflict heredity*:
$$\forall d, e, f \in E : d < e \wedge d \# f \Rightarrow e \# f,$$
- $l : E \rightarrow Act$ is a *labeling function.*

A prime event structure with binary conflict represents a concurrent system in the following way: action names $a \in Act$ represent actions the system might perform, an event $e \in E$ labeled with a represents an occurrence of a during a possible run of the system, $d < e$ means that d is a prerequisite for e and $d \# e$ means that d and e cannot happen both in the same run.

Causal independence (*concurrency*) of events is expressed by the derived relation $co \subseteq E \times E$: $d \; co \; e$ iff $\neg(d < e \vee e < d \vee d \# e)$. By definition, $<, >, \#$ and co form a partition of $E \times E$. Strict causal independence is expressed by $\smile \subseteq E \times E$: write $d \smile e$ iff $d \; co \; e \wedge d \neq e$.

$\mathbb{E}^{2\#}_{prime}$ denotes the domain of prime event structures with binary conflict labeled over arbitrary sets of actions. The components of an event structure $\mathrm{E} \in \mathbb{E}^{2\#}_{prime}$ will be denoted by $E_{\mathrm{E}}, <_{\mathrm{E}}, \#_{\mathrm{E}}$ and l_{E}. If clear from the context, the index E will be omitted. As usual, we write $d \leq e$ for $d < e \vee d = e$, etc.

Two event structures E and F in $\mathbb{E}^{2\#}_{prime}$ are *isomorphic*—notation $\mathrm{E} \cong \mathrm{F}$—iff there exists a bijection between their sets of events preserving $<, \#$ and labeling. Often one does not distinguish isomorphic event structures.

The behavior of an event structure is described by explaining which subsets of events constitute possible (partial) runs of the represented system (thus formalizing the interpretation of the conflict and the causality relation). These subsets are called *configurations*.

We will only consider finite configurations here. As usual, we assume that in a finite period of time only finitely many actions may be performed. Now the requirement says that we only consider runs which are executable in a finite period of time. This is no restriction since the infinite configurations which are usually considered are completely determined by the finite ones.

Definition 2 The set $C(\mathrm{E})$ of (finite) *configurations* of a prime event structure with binary conflict $\mathrm{E} = (E, <, \#, A, l)$ consists of those finite $x \subseteq E$ which are

- *conflict-free*: $\# \cap (x \times x) = \emptyset$,
- *left-closed*: $\forall d, e \in E: \; e \in x \wedge d < e \Rightarrow d \in x$.

Configurations may also be considered as possible states of the system; they determine the remaining behavior of the system as being the set of all events which have not yet occurred and are not excluded because of conflicts.

In graphical representations of prime event structures, only immediate conflicts — not the inherited conflicts — are indicated. The $<$-relation is represented by arcs, omitting those derivable by transitivity. Furthermore, instead of events only their labels are displayed; if a label occurs twice it represents two different events. Thus these pictures determine event structures only up to isomorphism.

Example 1 The system $(a \parallel b) + (a;b;c)$, executing either a and b independently or a, b and c sequentially, may be represented by the prime event structure with events e_1, e_2, e_3, e_4, e_5 with $l(e_1) = l(e_3) = a$, $l(e_2) = l(e_4) = b$ and $l(e_5) = c$, where e_1 *co* e_2, $e_3 < e_4$, $e_4 < e_5$, $e_3 < e_5$ and each of e_1, e_2 is in conflict with each of e_3, e_4, e_5. Following the conventions set out above, this event structure is represented as

$$\begin{array}{l} a \\ \# \\ a \rightarrow b \rightarrow c \quad . \\ \# \\ b \end{array}$$

Its configurations are $\emptyset, \{e_1\}, \{e_2\}, \{e_1, e_2\}, \{e_3\}, \{e_3, e_4\}, \{e_3, e_4, e_5\}$.

2 Process graphs

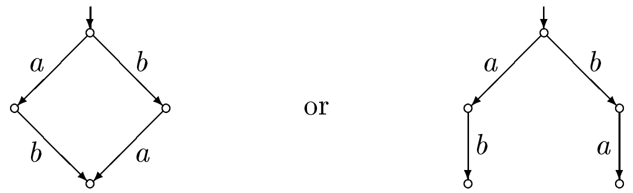
Definition 3 A *process graph (over an alphabet Act)* is a triple (S, T, I) where

- S is a set of *states* or *nodes*,
- $T \subseteq S \times Act \times S$, the *transitions* or *edges*,
- $I \in S$, the *initial state* or *root*.

Process graphs are also called *automata* or *state transition diagrams*. They model systems that are capable of performing actions from Act. The presence of a transition (p, a, q) indicates that the represented system can evolve from state p to state q by performing the action a. The domain of process graphs (over arbitrary alphabets) will be denoted $\mathbb{G}$. The components of a process graph $\mathrm{G} \in \mathbb{G}$ are denoted by S_{G}, T_{G} and I_{G} (a convention that also applies to other structures given as tuples). Moreover $p \xrightarrow{a}_{\mathrm{G}} q$ denotes the statement $(p,a,q) \in T_{\mathrm{G}}$. If $p \xrightarrow{a}_{\mathrm{G}} q$ then (p,a,q) is said to be an *outgoing edge* of p with *label* a. When clear from the context, the index G is omitted.

Two process graphs G and H are *isomorphic*—notation $\mathrm{G} \cong \mathrm{H}$—iff there is a bijective mapping between their sets of states that preserves T and I. Usually one does not distinguish isomorphic process graphs. This means that one abstracts from the identity of the nodes. In drawings of process graphs states are represented as dots, a transition (p,a,q) is represented by an a-labeled arrow from p to q and the initial state is represented by an incoming arrow (not originating from another state). Hence these drawings represent process graphs only up to isomorphism.

Process graphs are suited for representing nondeterministic systems while abstracting from their real-time behavior and stochastic aspects. They are often used in interleaving semantics where parallel composition is modeled as arbitrary interleaving [Mil89, BW90]. In interleaving semantics the systems $a \parallel b$ (performing the actions a and b independently in parallel) and $ab+ba$ (performing either first a and then b or first b and then a) can each be represented by either



which are considered semantically equivalent. As a consequence, information about causal dependence is lost in this model. Now the idea behind *history preserving process graphs* [Gla88] is simple: squares represent concurrency and branching represents choice or conflict. Moreover, opposite edges in a square represent the same occurrence of an action. So $a \parallel b$ is represented by the graph on the left and $ab+ba$ is represented by the graph on the right. These graphs are no longer equivalent. It turns out that this simple feature makes process graphs capture all causal subtleties that are expressible by means of event structures, and more.

3 Translating between event structures and process graphs

As configurations model states of event structures, it is trivial to translate labeled prime event structures into automata.

Definition 4 The process graph $\mathcal{G}(\mathrm{E})$ *associated* to an event structure E is $(C(\mathrm{E}), T_{\mathrm{E}}, \emptyset)$, where $(x,a,y) \in T_{\mathrm{E}}$ iff $x \subset y$, $|y-x|=1$ and a is the label of the unique event in $y-x$.

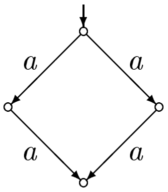
In order to translate process graphs with back into labeled prime event structures I formalize the postulate that opposite edges in a consistently labeled square represent the same action occurrence.

Definition 5 Let $\mathrm{G} \in \mathbb{G}$. Then $\sim$ is the smallest equivalence relation on T_{G} satisfying

$$p \xrightarrow{a} q \xrightarrow{b} s,\ p \xrightarrow{b} r \xrightarrow{a} s,\ (a \neq b \vee q \neq r)\ \Rightarrow\ (p,a,q) \sim (r,a,s).$$

If $u \sim v$ then the edges u and v are said to *represent the same event*. The equivalence classes are called *events* of G. Let E_{G} be the set of events of G. $[p,a,q]$ denotes the event containing (p,a,q).

In a history preserving process graph an event denotes an occurrence of an action in a possible run of the represented system. It thus represents a level of abstraction between actions and transitions. The graph on the right for instance has 4 transitions, grouped together into 2 events, in turn grouped together into 1 action. Events can be thought of as the basic building blocks of the represented system. An action then represents an equivalence class of events that are indistinguishable on the chosen level of abstraction at which the system is represented. A transition, on the other hand, represents an event as it can be totally ordered in time w.r.t. other events occurring in the same run. As in reality it may be meaningless to order independent events in time (they could happen lightyears apart in space and happen more or less simultaneously), transitions are not so fundamental as events; they are merely a convenience in the representation of a system as a process graph.



Definition 6 A (finite) *rooted path* π in a process graph G is a sequence of edges of the form $I_{\mathrm{G}} \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} p_n$.

As an edge determines an event, a rooted path in a process graph determines a sequence of events that can be performed by the represented system. Since some of the events in such a sequence may be causally independent, and the order of these events is semantically irrelevant, it makes sense to consider the multiset representation of such a sequence. This is called a *configuration*.

Definition 7 A *configuration* of a process graph G is a multiset $\Sigma_{i=1}^{n}[p_{i-1}, a_i, p_i]$ determined by a rooted path $(I_{\mathrm{G}}, a_1, p_1)(p_1, a_2, p_2)(p_2, a_3, p_3)\cdots(p_{n-1}, a_n, p)$ in G.

Process graphs of the form $\mathcal{G}(\mathrm{E})$ are acyclic, and all their configurations are plain sets. For such graphs the event structure E can be easily reconstructed.

Definition 8 The labeled prime event structure with binary conflict $\mathcal{E}(\mathrm{G})$ *associated* to a process graph G is $(E_{\mathrm{G}}, <_{\mathrm{G}}, \#_{\mathrm{G}}, l_{\mathrm{G}})$, where

- E_{G} is given by Definition 5,
- $d <_{\mathrm{G}} e$ if every configuration that contains d also contains e,
- $d \#_{\mathrm{G}} e$ if their is no configuration containing both d and e,
- and $l_{\mathrm{G}}([p, a, q]) = a$.

Note that $(p, a, q) \sim (p', a', q') \Rightarrow a = a'$; hence the definition of $l_{\mathrm{G}}(e)$ above is independent of the choice of a transition (p, a, q) representing e. Now I can state the main theorem of this handout:

Theorem 1 Let E be a labeled prime event structure with binary conflict. Then $\mathcal{E}(\mathcal{G}(\mathrm{E})) \cong \mathrm{E}$.

Homework

1. Give an example of an event structure in which $\#$, $<$ and $\smile$ are nonempty relations; translate this structure in a process graph using Definition 4 and translate the result back in an event structure using Definition 8. Compare the endresult with your original event structure.

2. Given a process graph, a tree-representation of the same system can be obtained by means of *unfolding*. Give a formal definition of the unfolding $\mathcal{U}(\mathrm{G})$ of a process graph G.

3. Last week we have shown that trees contain less information than event structures, i.e. after translating an event structure into a tree we can't always recover the original event structure (working up to isomorphism). A process graph is in general a smaller object than the corresponding tree, as some of its branches are shared. Yet, process graphs contain the same information as event structures (by Theorem 1), and hence less than trees. How can that be possible?

4. If we start with a process graph, translate into an event structure, and translate the result back into a process graph, do we always get back the original graph? Explain (briefly).

5. * Prove the theorem above.

References

[BW90] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press.

[Gla88] R.J. van Glabbeek (1988): *An operational non-interleaved process graph semantics of CCSP* (abstract). In E.-R. Olderog, U. Goltz & R.J. van Glabbeek, editors: *Combining compositionality and concurrency, summary of a GMD-workshop, Königswinter, March 1988*, Arbeitspapiere der GMD 320, pp. 18–19.

[Mil89] R. Milner (1989): *Communication and Concurrency.* Prentice Hall, Englewood Cliffs.

[NPW81] M. Nielsen, G.D. Plotkin & G. Winskel (1981): *Petri nets, event structures and domains, part I. Theoretical Computer Science* 13(1), pp. 85–108.